ᴄᴘX ATARI® PROGRAM EXCHANGE

HARRY STEWART   MARCH 1982

EXTENDED WSFN

<u>CASSETTE</u> (APX-10026)
REQUIRES: 16K RAM

<u>DISKETTE</u> (APX-20026)
REQUIRES: 24K RAM

User-Written Software for ATARI Home Computers

HARRY STEWART   MARCH 1982

EXTENDED WSFN

CASSETTE (APX-10026)
REQUIRES: 16K RAM

DISKETTE (APX-20026)
REQUIRES: 24K RAM

EXTENDED WSFN

by

Harry Stewart

Program and Manual Contents © 1982 ATARI, Inc.

TRADEMARKS OF ATARI

The following are trademarks of Atari, Inc.

ATARI®
ATARI 400™ Home Computer
ATARI 800™ Home Computer
ATARI 410™ Program Recorder
ATARI 810™ Disk Drive
ATARI 820™ 40-Column Printer
ATARI 822™ Thermal Printer
ATARI 825™ 80-Column Printer
ATARI 830™ Acoustic Modem
ATARI 850™ Interface Module

IMPORTANT!

DUPLICATE
THIS
DISKETTE
BEFORE
USING
THIS
PROGRAM!
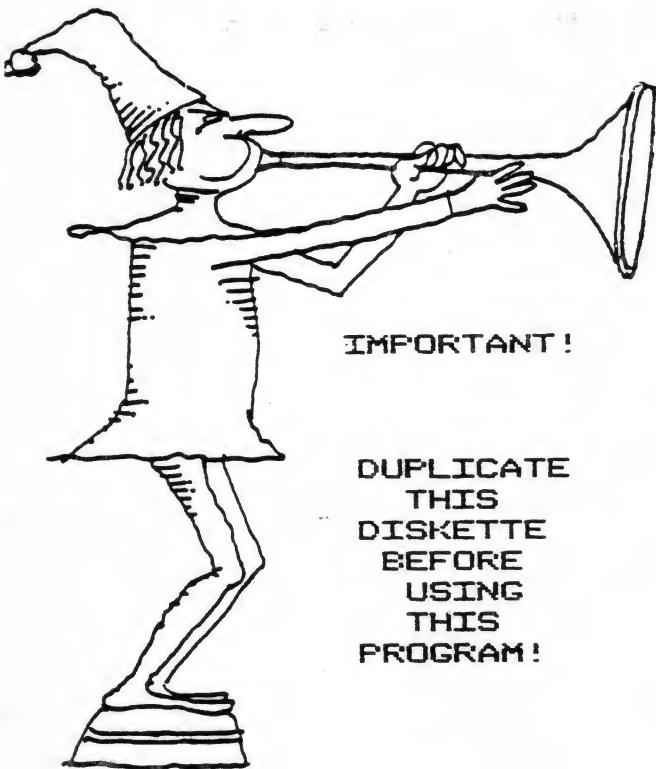
This APX diskette is unnotched to protect the software against
accidental erasure. However, this protection also prevents a program
from storing information on the diskette. The program you've
purchased involves storing information. Therefore, before you can use
the program, you must duplicate the contents of the diskette onto a
notched diskette that doesn't have a write-protect tab covering the
notch.

To duplicate the diskette, call the Disk Operating System (DOS) menu
and select option J, Duplicate Disk. You can use this option with a
single disk drive by manually swapping source (the APX diskette) and
destination (a notched diskette) until the duplication process is
complete. You can also use this option with multiple disk drive
systems by inserting source and destination diskettes in two separate
drives and letting the duplication process proceed automatically.
(Note. This option copies sector by sector. Therefore, when the
duplication is complete, any files previously stored on the
destination diskette will have been destroyed.)

PREFACE

WSFN (TURTLE) was originally planned for release in cartridge form. This version is either on diskette or cassette. Therefore, please substitute the word applying to the version you have whenever you read the word "cartridge" in these instructions.

Two documents accompany TURTLE. ATARI WSFN (TURTLE) is a tutorial. EXTENDED WSFN (TURTLE) is a reference manual.

REQUIRED ACCESSORIES

    16K RAM for cassette version
    24K RAM for diskette version
    ATARI 410 Program Recorder for cassette
    ATARI 810 Disk Drive for diskette

GETTING STARTED

If you have the cassette version of TURTLE:

1. Have your computer turned OFF.

2. Insert the TURTLE cassette in the program recorder, press REWIND, and then press PLAY.

3. Turn on your computer while holding down the START key.

4. When you hear the beep, release the START key and press the RETURN key. TURTLE will load into RAM and you'll see the TURTLE display screen.

If you have the diskette version of TURTLE:

1. Turn on your disk drive, insert the TURTLE diskette, and power up your computer.

2. When the READY prompt displays, type DOS to call up the menu.

3. Enter menu selection L (Binary Load).

4. To the LOAD FROM WHAT FILE? prompt, enter TURTLE and press the RETURN key. TURTLE will load into RAM and you'll see the TURTLE display screen.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

1. TURTLE OVERVIEW

The turtle package provides the user with the ability to generate screen graphics using a few simple keyboard commands. The drawing element is called a turtle, which may move around the screen leaving "tracks" (or not, at the user's discretion). His motion and direction are controlled with five commands, each of which is a single keystroke:

    H - Place turtle on home position (center of screen).

    N - Point turtle north (up).

    R - Rotate turtle 45° clockwise.

    L - Rotate turtle 45° counterclockwise.

    F - Move turtle forward one unit.

These commands are sufficient to draw pictures of any degree of complexity, when coupled with other commands and capabilities. Beyond the command set provided by the turtle package, the user may define additional commands which are conglomerations of other system or user commands. Nested and/or recursive command structures may be defined which behave very much like programs in more procedure-oriented languages.

1.1 TURTLE'S DOMAIN

The turtle's domain consists of two regions: (1) the visible region and (2) the invisible region. The domain is fixed in size as a 65536 by 65536 "sphere" with the visible region being variable in size, depending on the current display mode and resolution. The turtle may be confined to the visible region or not, at the user's discretion.
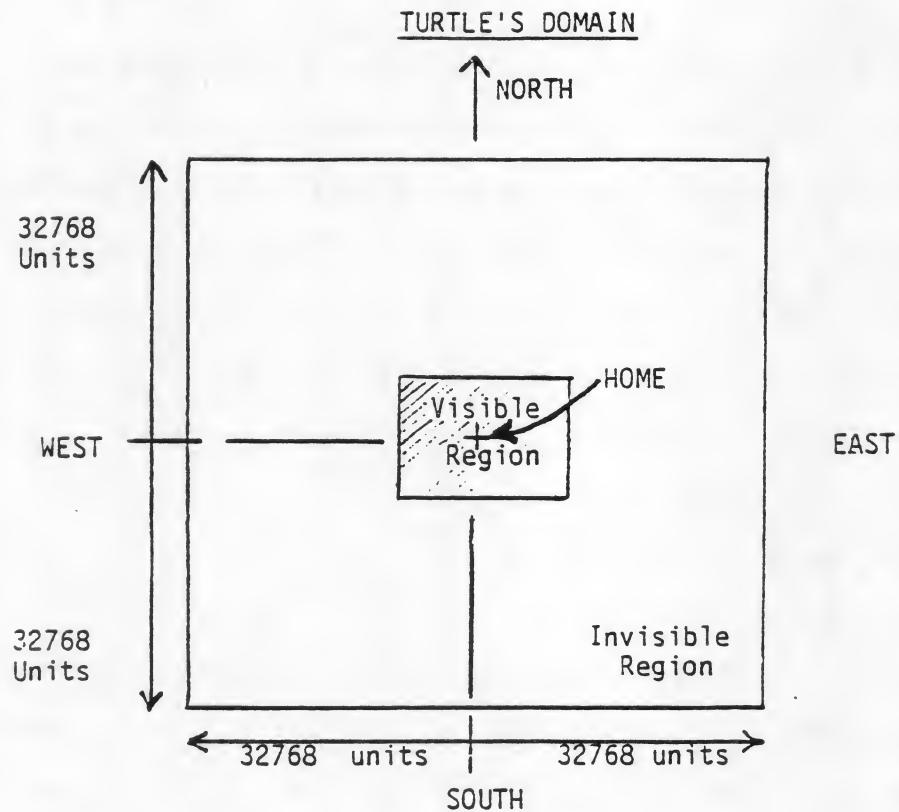
TURTLE'S DOMAIN

Note that the domain is "spherical" in that the East & West
edges are joined as are the North & South edges.

The turtle is always defined by his position and orientation. From
his position, he may move to any of 8 adjacent "cells" based upon his
orientation, as shown below.
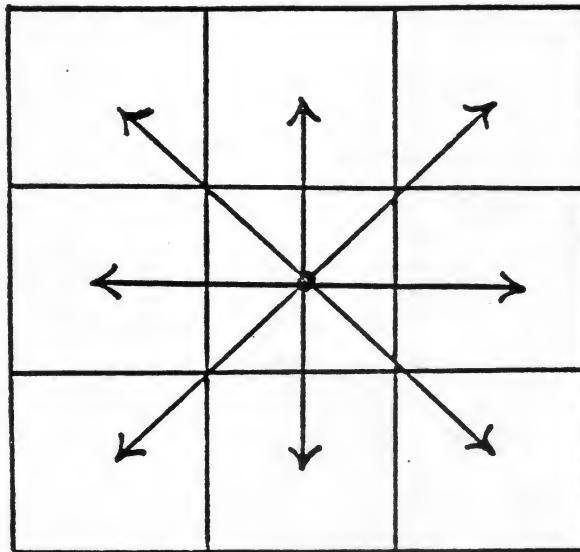
In addition to moving to an adjacent cell, the turtle may also "sense"
whether a cell is in the visible region or not, and detect the presence
of a "track" in an adjacent cell.

1.2 DRAWING

The turtle draws by leaving "tracks" with a magic marker which is con-
trolled with three commands.

     U--which stands for pen up; this allows the turtle to move without
         leaving tracks.

     P--is the pen select command; this selects one of three colors for
         the pen, or erase (which removes tracks that the turtle crosses).

     D--which stands for pen down; this causes the turtle to leave
         tracks of the current pen color.

The turtle can also detect tracks that are already in his region with the "S" (sense) command.

1.3 CONTROL OF THE TURTLE

The turtle is controlled by keyboard commands, which are executed as they are entered. Command execution may be deferred, however, by using the nesting capability provided by the () and [ ] matched sets. Thus, if (CHNR999F) is entered, none of the commands within the parens will be exe- cuted until the matching right paren is entered--at which time the string of commands will be executed at full speed. The command string above does the following:

> C - Clear the screen
>
> H - Home the turtle
>
> N - Face him North
>
> R - Rotate him to the right (now facing NE)
>
> 999F - Move forward 999 times (each time leaving a track, if
>
> > the pen is down).

This example illustrates another feature, that of repetition (or iteration); a number preceding a command indicates that the command is to be executed as many times as indicated by the number. This feature also applies to the nesting brackets; thus, the example below includes nested iterations:

<p align="center">4(2R 4(1OF2R))</p>

<p align="center">Draws a square</p>

Draws 4 squares with a common corner:
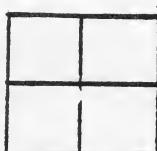
1.4 KEYBOARD CONTROL

The keyboard is monitored at all times, even while a command is being exe-
cuted; if a key is pressed, the current command is temporarily suspended,
the newly entered command is executed, and then the prior command is re-
sumed. This command interruption may actually be accomplished to any num-
ber of levels (commands interrupting commands interrupting commands, etc.).

The BREAK key will stop all command activity and put the turtle into an
idle state.

1.5 USER DEFINED COMMANDS

The user may define unused keys to be "macro" commands--conglomerates of
intrinsic and/or user defined commands. These commands may recurse, if
desired, and may also, in turn, define other commands or redefine themselves.

2. COMMAND FORMS

Turtle commands fall into one of six forms, as will be described in the paragraphs that follow.

2.1 SINGLE LETTER

```
<command> ::=  <name>
```

Many commands are of this form, where a <u>single</u> key stroke (the name) is the command. Examples are: H (Home), C (Clear), F (Forward).

2.2 TESTING COMMANDS

```
<command> ::= <name>  <then>  <else>
```

There are four commands of this form, where a test is made and <u>either</u> the <then> clause or the <else> clause is executed, (but not both). Examples are: T (Test Acc.), E (Edge test) and ? (Random test).

2.3 ITERATION

```
<command> ::= <value>  <clause>
```

There are three commands of this form, where a numeric value is used to iterate a following clause. Examples are: A (Iterate by ACC), # v (Iterate by variable), and n (Iterate by number).

2.4 NESTING

```
<command> ::= <nesting bracket><clause>..<clause><matching bracket>
```

There are two nesting bracket sets: (...) and [...] .

2.5 COMMAND/VALUE

```
<command> ::= <name> <value>
```

2.6 DEFINITION

```
<command> ::= <name> [*] <clause>
          ::= <name> #
```

3.   SPECIFIC COMMANDS

   3.1   TURTLE POSITION & ORIENTATION

      - Home Turtle - | H |

        Puts turtle to the center of the screen and leaves his "track"

        if the pen is down; does not alter turtle orientation.

      - Point Turtle North - | N |

        Faces the turtle towards the top of the screen without altering

        his position.

      - Turtle Forward - | F |

        Moves the turtle forward one unit and leaves his "track" if the

        pen is down; does not alter turtle orientation unless the screen

        edge is hit in reflect mode.

      - Rotate Turtle Right - | R |

        Rotates the turtle clockwise 45°.

      - Rotate Turtle Left - | L |

        Rotates the turtle counterclockwise 45°.

   3.2   TURTLE "SENSES"

      - Sense Color Value - | S |

        Reads the color of the square in front of the turtle and puts it

        into the turtle accumulator (ACC). Background is read as zero,

        turtle tracks are read just as they were written, and squares

        beyond the screen edge are read as zero.

3.2 TURTLE "SENSES" (continued)

- Joystick/Trigger Sense - | $ <select> <on> <off> |

Tests the selected joystick position, joystick trigger and
executes either the <on> clause or <off> clause, based
upon the result of the test. See Appendix F for a list of
the select codes for the various sensible items.

Joystick position and triggers are continuously sensed;
there is no edge detection or reset logic.

- Pot Controller Read - | % <select> |

Reads the selected pot controller and puts its value in the
accumulator; the controller range is from Ø (full counter-
clockwise) to 228 (full clockwise). The <select> values
range from Ø to 7 corresponding to the 8 pot controllers.

- Orientation Sense - | ; |

Sets the accumulator to one of the values shown below, based
on the current orientation:

Ø - N

1 - NE

2 - E

3 - SE

4 - S

5 - SW

6 - W

7 - NW
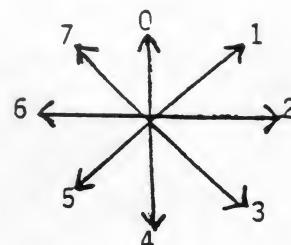
3.2 TURTLE "SENSES" (continued)

- Edge Test - | E <true> <false> |

Tests to see if the square in front of the turtle is at or beyond
the screen edge; if so, the <true> clause is executed; otherwise,
the <false> clause is executed.

3.3 TURTLE MANIFESTATIONS

- Beep - | B |

Generates an audible tone using the television sound system.

- Pen Up - | U |

Allows the turtle to move without leaving tracks; this command
is countermanded by the "D" command described below.

- Pen Down - | D |

Lowers the pen, which means that the turtle will leave tracks
(or erase), depending upon the current pen color selected.

- Audio Control - | CTRL-A <letter> |

Allows sounds to be generated as manifestations of internal
registers of the turtle program, as shown in Appendix H.

- Wait - | W |

Causes the turtle to wait for the next 30 HZ clock tick before
resuming execution of turtle commands. See also 3.10 for a
description of the speed control command.

3.3 TURTLE MANIFESTATIONS (continued)

- Pen Select - | P |

The value of the turtle accumulator modulo 128 is used as
the color select for turtle tracks, until changed. Zero is
background (erase) and is not the same as issuing a "PEN UP"
command.

- Turtle Representation - | CTRL-T ⟨number⟩ |

Selects or de-selects a turtle overlay that nondestructively
shows turtle position and orientation at all times.

    ⟨number⟩ = 0, de-selects the overlay;

    ⟨number⟩ = 1, selects an arrow overlay;

    ⟨number⟩ = 2, selects a turtle overlay;

    ⟨number⟩ = 3, selects a point overlay.

3.4 TURTLE WORLD

- Clear Screen - | C |

Clears all turtle tracks from the screen without altering the
turtle position or orientation.

- Edge Rule Select - | CTRL-E ⟨number⟩ |

Selects one of four rules to follow when the turtle encounters
the edge of the screen.

    number = 0, turtle stops;

    number = 1, turtle wraps to opposing edge;

    number = 2, turtle bounces (reflects) off edge;

    number = 3, turtle goes off edge but does not leave
                            tracks until he gets back in screen boundary.

3.4 TURTLE WORLD (continued)

If the turtle is not in the screen boundaries when an edge
rule is selected, he will be put to the home position.

- Display Mode Select   - | CTRL-D <number> |

Preselects one of the eight display modes available (see Appen-
dix C for descriptions of each mode). The mode change takes
place with the next operating mode select (CTRL-M, see 3.10).

- Color Register Values   - | & <select> |

Allows the hardware color register selected to be updated with
the value in the turtle accumulator. See Appendix D for the
utilization of the color registers for each screen mode and
see Appendix E for color register values to use to get the
desired colors and luminescence.

3.5 TURTLE ARITHMETIC

- Increment Turtle Accumulator   - | + |

Adds one to the four-digit ACC; will not increment it past all
nines.

- Decrement Turtle Accumulator   - | − |

Subtracts one from the four-digit ACC; will not decrement it
past zero.

- Test ACC for ≠ Zero   - | T <true><false> |

Tests to see if the ACC is greater than zero; if so, the <true>
clause is executed; otherwise, the <false> clause is executed.

- Set ACC to Zero·   - | @ |

Sets the ACC to zero.

3.6   ITERATION

- Iterate by Constant   -   `⟨constant⟩ ⟨clause⟩`

Iterates the ⟨clause⟩ by the number indicated in the constant.
If the constant is zero, the ⟨clause⟩ is not executed at all.
If ⟨constant⟩ exceeds four digits in length, the last four
digits are used.

- Iterate by Variable   -   `# ⟨variable⟩ ⟨clause⟩`

Same as above, except current value of the indicated variable
is used instead of a constant.

- Iterate by Turtle Accumulator   -   `A ⟨clause⟩`

Same as for CONSTANT, except current value of the ACC is used
instead of a constant. The ⟨clause⟩ may modify the ACC without
changing the iteration count.

- Stop Iteration   -   `!`

When executed, makes the current iteration the last iteration
(set the iteration count to zero) within the current iteration
level. This command will not affect outer level iteration counts.

- Increment Iteration Count   -   `∧`

When executed, increments the current iteration count. Has
no effect if not executed within an iteration.

3.7   NESTING COMMANDS & CLAUSES

- Basic Nest   -   (⟨clause⟩ ... ⟨clause⟩)

Any number (including zero) of ⟨clauses⟩ may be nested together
creating a single new clause, using matched parentheses. Parens
may exist within other parens to any level desired.

3.7   NESTING   COMMANDS & CLAUSES (Continued)

- Accumulator Save/Restore Nest -   [...]

Behaves the same as above except that the turtle accumulator

is saved and restored by the [and] commands.

3.8   USER DEFINED COMMANDS & VARIABLES

- Define User Command - | = <name> <clause>
                          | = *<name> <clause>

Creates a definition in memory whereby the < clause > may be

invoked merely be using the <name>; the optional * allows

<name> to be the same as one of the turtle intrinsic commands.

If the < clause > consists of a blank character (NOP), the <name>

will be removed from the user command directory.

- Invoke User Command - | <name>
                          | *<name>

Once defined, a user command may be used exactly as an intrinsic

command is used; however, if <name> is the same as an intrinsic

command, the * must be used, as intrinsic commands have a higher

priority in the name search.

- Get User Command Definitions - | CTRL-G " <name> "

This command clears the current set of user commands and reads in

a new set from the indicated device. The device name is in the

standard format, e.g., "C:", "D:HILBERT", etc.

- Put User Command Definitions - | CTRL-P " <name> "

This command writes the current set of user commands to the indi-

cated device. The device name is in the standard format, e.g.,

"C:", "P:", "D:HILBERT", etc.

3.8 USER DEFINED COMMANDS & VARIABLES

- Load ROM Resident Command Definitions - | CTRL-L <character> |

This command clears the current set of user commands and reads
a new set from ROM. The character following the command indi-
cates which of several sets to load. See Appendix G for a list
of command sets and their content.

- Run ROM Resident Command - | CTRL-R <character> |

This command performs all of the functions of the LOAD command
described above, and in addition, executes one of the commands
loaded. See Appendix G for more information.

- Clear User Variables - | CTRL-C |

Removes all user variables from memory.

- Define User Variable - | = # <name> |

Creates a definition in memory whereby the current value of the
turtle accumulator is assigned to the indicated <name> . Any
character may be used as a <name>. See section 3.6 for the
use of variables.

3.9 RANDOM TEST

- <u>Random Test</u> - | ? $\langle$then$\rangle$$\langle$else$\rangle$ |

Randomly executes either the $\langle$then$\rangle$ or the $\langle$else$\rangle$ clause,

using a hardware random bit generator.

3.10 MODE CONTROL & OPTIONS

- <u>Edge Rule</u> - | CTRL-E $\langle$number$\rangle$ |

CTRL-E Ø = Stop at edge;

CTRL-E 1 = Wrap at edge;

CTRL-E 2 = Reflect at edge ;

CTRL-E 3 = Disappear at edge.[*]

- <u>Speed Control</u> - | CTRL-S $\langle$number$\rangle$ |

CTRL-S Ø = Run full speed[*];

CTRL-S 1 = Single step (press CTRL-4 to step);

CTRL-S 2-7 = Run with delays:

2 = 30 commands/sec.

3 = 15 commands/sec.
:
7 $\approx$ 1 command/sec.

---

[*] power-up default

3.10   MODE CONTROL & OPTIONS (continued)

- <u>Operating Mode</u>   - | CTRL-M <number> |

CTRL-M Ø =   Draw mode (full screen graphics) ;

CTRL-M 1 =   Debug mode (full screen text);

CTRL-M 2 =   Split screen with internal registers;

CTRL-M 3 =   Normal mode (Split screen with input echo only).*

<u>DRAW MODE</u>: In draw mode, the entire screen is dedicated to turtle graphics. Commands are executed as they are entered and the user is typing "blind" (the commands are executed, but there is no echoing of the command name to the screen).

<u>DEBUG MODE</u>: In debug mode, the entire screen is dedicated to text data. Commands to be executed must be entered using the RETURN KEY and line editing may be performed upon input command lines. The screen shows the internal workings of the turtle and also shows all user defined variables and command definitions.

<u>SPLIT-DEBUG MODE</u>: In split-debug mode, the upper portion of the screen is dedicated to turtle graphics and the lower portion contains four text lines. The first two text lines show the turtle registers as in debug mode and the other two text lines are for command entry.

---

* power-up default

NORMAL MODE: Normal mode is similar to split-debug except
that the text portion of the screen is used solely for com-
mand entry. When one complete command has been entered, the
cursor will move to the beginning of the next line (this may
cause the text to scroll); when the command has finished the
execution, the cursor will move to the beginning of the next
line.

DRAW MODE SCREEN (MODE Ø):

Graphics data
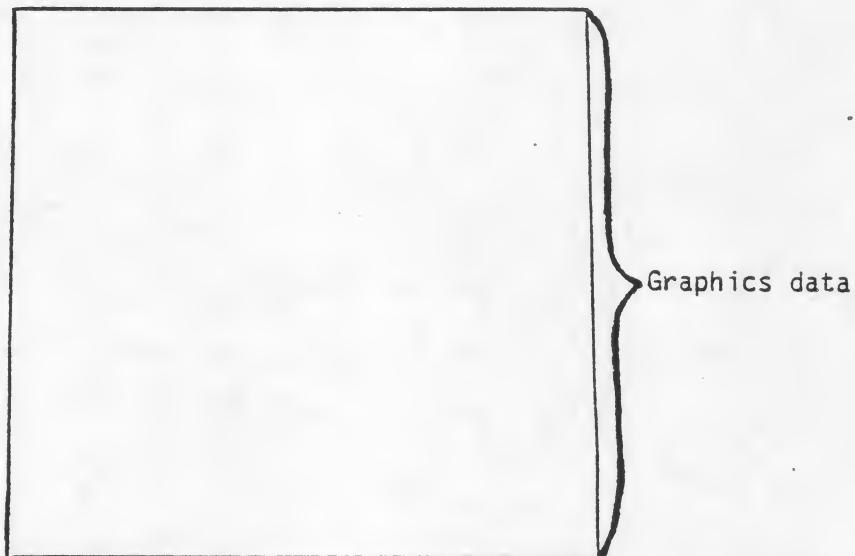
Turtle accepts inputs as typed without echoing them to the
screen; no line editing is allowed.

DEBUG MODE SCREEN   (MODE 1)

(1)   ACC=0000   NUMBER=0000   LEVEL=0000 } Execution data
       CHAR=F   ERROR=

(2)   >   } User input line

       (CHNA-44F)   } Previous command line

       A=0001   X=0022   Z=0005   } User defined variables

       G=4F
       I=4(1OFZR)
       Z=A-
                                              } User defined commands
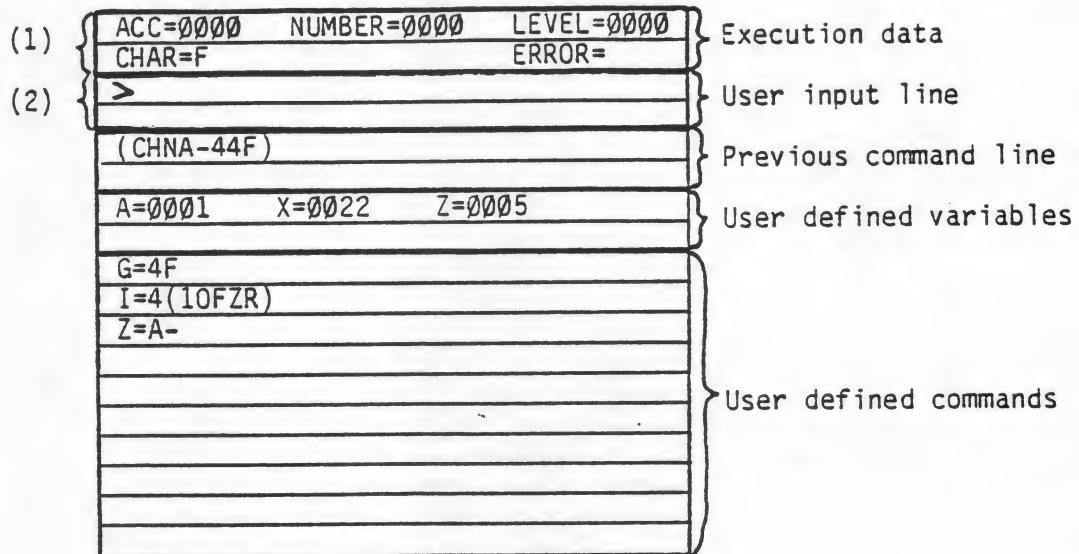
(1)   ACC   =   Turtle accumulator value

       NUMBER =   Iteration counter value

       LEVEL   =   User defined command "call" level

       CHAR   =   Current (or last) executed command

       ERROR   =   Command error code (see Appendix B)

(2)   Turtle accepts inputs when ">" is present; ">" disappears

       while a command is executing. Input lines must be termin-

       ated by the RETURN key; line editing is allowed.

SPLIT-DEBUG MODE SCREEN   (MODE 2)

```
        ┌─────────────────────────────┐
        │                             │ ⎫
        │                             │ ⎬  Graphics data
        │                             │ ⎭
(1)     │ ACC=0000   NUMBER=0000   LEVEL=0000 │ ⎫ Execution data
        │ CHAR=F                ERROR=  │ ⎭
(2)  ⎨  │ >                           │    User input line
        └─────────────────────────────┘
```
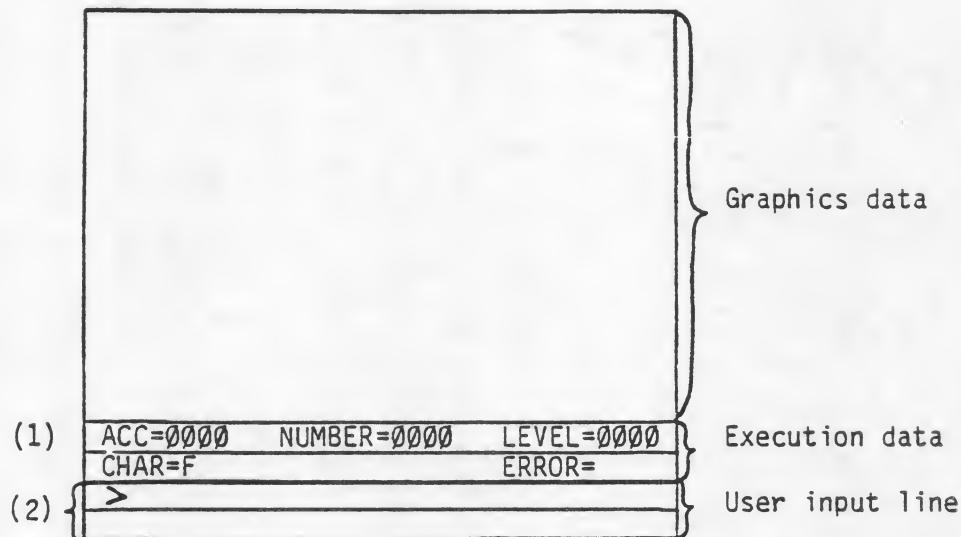
(1) See prior page for explanation.

(2) Turtle accepts inputs when ">" is present; ">" disappears
     while a command is executing. Commands are executed as
     input and no line editing is allowed.

NORMAL MODE SCREEN   (MODE 3)

} Graphics data

(1) } User input area
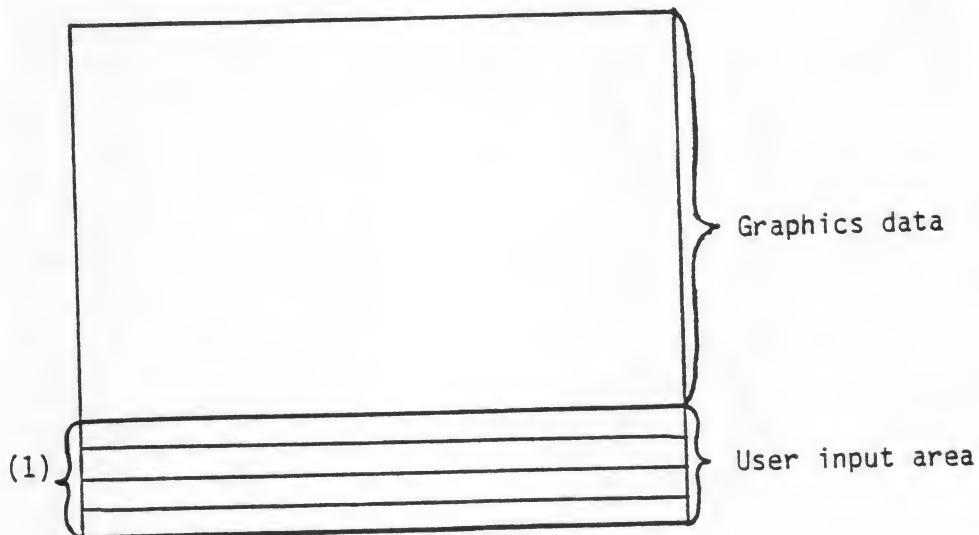
(1)   Commands are scanned as input and no line editing is
      allowed. The cursor moves to the beginning of the
      line below the input line (or the input line srolls
      upward) when a command is accepted and does the same
      again when command execution is complete.

- Display Mode - | CTRL-D <number> |

      CTRL-D Ø - 7   = Map to O.S. display modes
                          1-8, where higher numbers
                          represent increasingly
                          higher resolution display.

The system defaults to CTRL-D 6 (Display Mode 7).
This command acts as a pre-select; the mode change actually
occurs at the next CTRL-M command.

3.11 NO-OPS

      Blank, underscore, and all unassigned keys are treated as no-
      operation codes.

4.   EXAMPLES

    4.1   Set Acc = $\emptyset$

                ┌─────────┐
                │   A-   │
                └─────────┘

    4.1   Set Acc = 23

                ┌─────────┐
                │ A-23+ │
                └─────────┘

    4.2   Set Acc = Acc * 2

                ┌─────────┐
                │   A+   │
                └─────────┘

    4.3   Set Acc = Acc * 7

                ┌─────────┐
                │ A(6+) │
                └─────────┘

    4.4   Set Acc = Acc / 7

                ┌───────────────┐
                │ =ZT(7-Z+)_   │
                │   (+Z-)   │
                └───────────────┘

    4.5   Set pen color to 1 without altering Acc

                ┌───────────┐
                │ [A-+P]   │
                └───────────┘

    4.6   Find upper right corner of screen

                ┌───────────────┐
                │ =XE_(FX)   │
                │   (UN̄X2RX D) │
                └───────────────┘

    4.7   Recurse by number in Acc

                ┌───────────────┐
                │ =ZT(-Z+)_   │
                │ Z   │
                └───────────────┘

4.8 Sierpinski Curve

```
=IT(-I2FI3LG3LI2FI+)2R
=G4F
=Y(HNU44F2R44FRC[A-+P]4(2FI))
```

After these are defined, one sets the accumulator to the
desired order and types "Y" to draw the curve. Again, "G"
can be redefined to a smaller number of "F"s to draw the
higher order "Y"s. For "DGOF," the accumulator can be
set to 5.

The Sierpinski curve is closed and consists of four iden-
tical sides arranged in different directions and connected
by a short line "2F." The sides are defined (recursively
again) in the macro "I" and the closed curve itself is de-
fined in the last part of the macro "Y," namely: "4(2FI)."
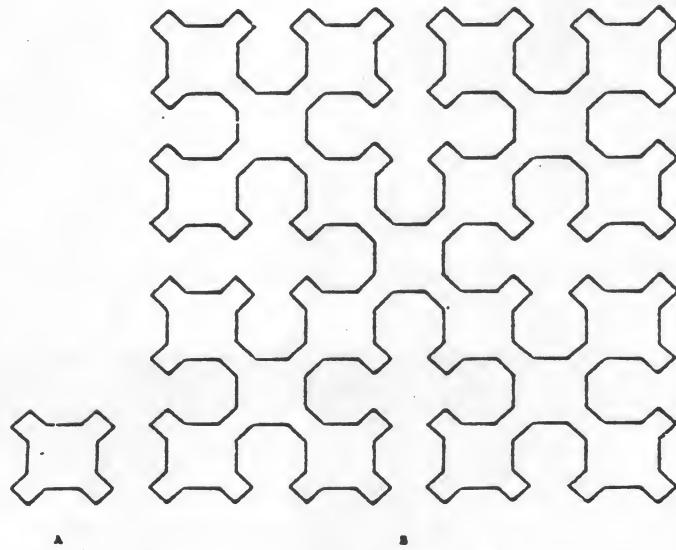
Sierpinski curve of (A) order 1, and (B) order 3.

4.9 Hilbert Curve

```
=ZT(-VG2LZ2RGZG2LV+)2L
=VT(-Z2RGVG2LVZRGZ+)2R
=G4F
 =J(HNU44F2R44FC2RDZ)
```

After these are defined, one sets the accumulator to the
desired order and types "J" to draw the curve. "G" can
be redefined to a smaller number of "F"s to draw higher
order curves.
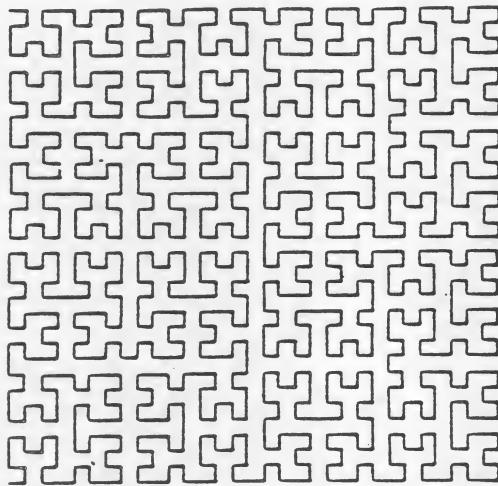
Hilbert curve of (A) order 1. and (B) order 5.

4.10   Trinary Tree Structure

```
=G(-T(++Z-AF2R3(G2R)AFA+)(+4R))
=ZT(2-Z+)
=J(CHNA-4∓99(4(G2R)A+))
```

Type "J" to Start drawing; press the BREAK key to stop.

origin
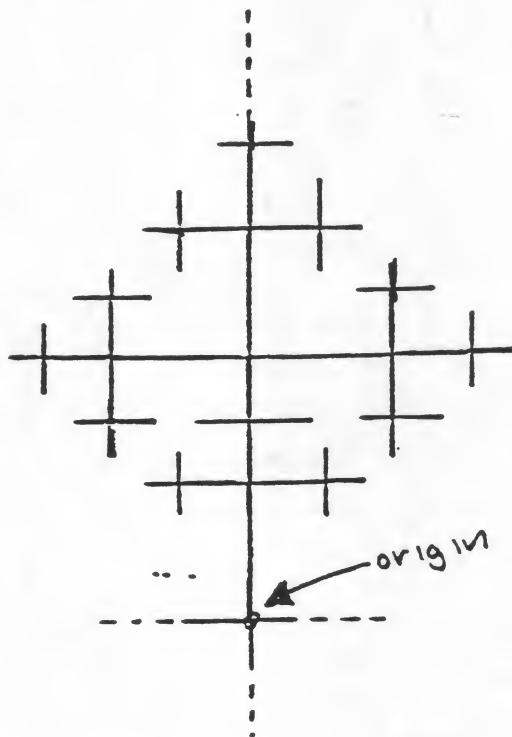
4.11   Spirals

```
=I(T(--2(2LAF)Z)_)
=V(++2(2RAF))
  =J[#IY4RU2FD4RZ]
```

To run: Set accumulator = Ø. (Type "A-").

          Set #I to spiral iteration count.

          (Type   "[n+=#I]"   -- try n=13 first)

          Type "Q".

4.11 Spirals (continued)
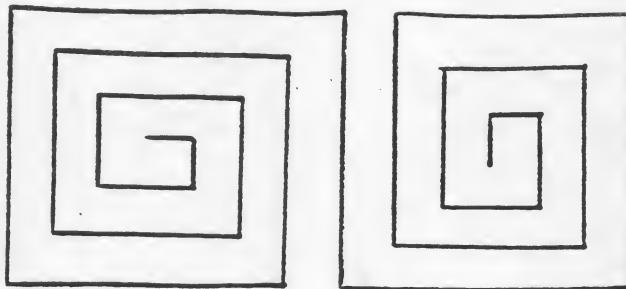
#I = 7

4.12 Super Spiral

```
=Q (2RIT(LIT(LIT(LIT(LITZ_)_)_)_)_)
=I (UFA-2L5TR(SR)RTFD4R)
=Z (B2LY)--initiate backtrack
=Y (2RJT (LJT(LJT(LJT(LJT5B_)_)_)_)_)
=J (ST (FY)I)
```

To run: Put obstacles on screen.

　　　　　Type (HN9999Q)

The command will produce a clockwise spiral figure which
avoids all obstacles in its path. It is basically an edge
follower with a backtrack algorithm which is invoked   when
there are no forward moves possible.

4.13 Diagnonal Plaid

```
(NR999(10F+P))
```

4.14   Random Pattern #1

```
(100(8?R2F10F))
```

4.15   Random Pattern #2

```
999?RF
```

4.16   Random Pattern #3

```
100(8?2R_10?F_)
```

4.17   A Koch Curve

```
=ZT(-ZG4L3(2RGZG)3(GZG2L)4RGZ+)_
=G2F
=J4(GZG2R)
```

After those are defined, one sets the accumulator to the desired
order and types "J" to draw the curve. "G" can be redefined to
a smaller number of "F"s to draw higher order curves.

A = 0

A = 1
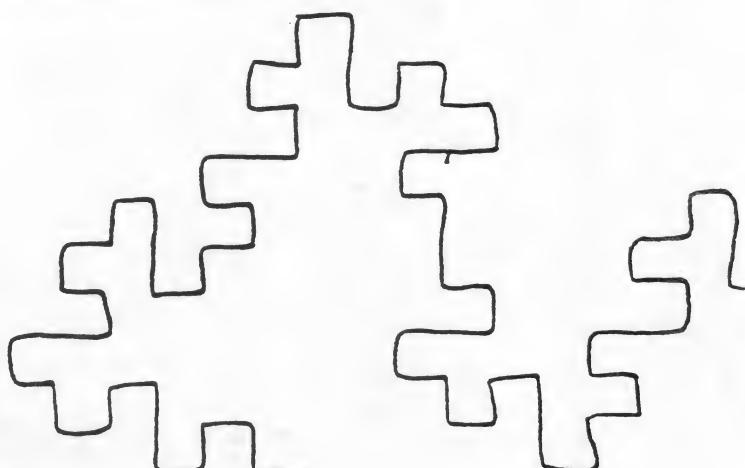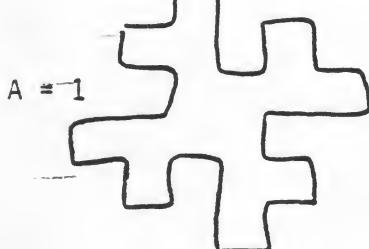
A = 2 (part of)
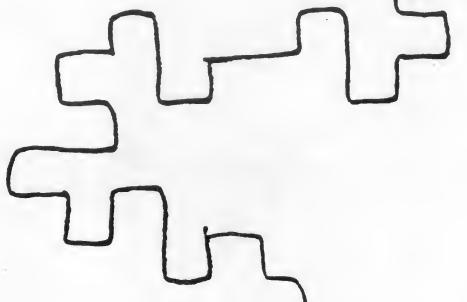
Hilbert Curve (written in Pascal)

```
PROGRAM HILBERT;
VAR SIZE,DELTA,N:INTEGER;
  ORDER:INTEGER;
  PROCEDURE HIL(I:INTEGER);
  VAR A,B:INTEGER;
    PROCEDURE HIL1;
    BEGIN
      TURN(A);  HIL(-B);  TURN(A);
    END (*HIL1*);
    PROCEDURE HIL2;
    BEGIN
      MOVE(SIZE);
      HIL(B);
      TURN(-A);  MOVE(SIZE);  TURN(-A);
      HIL(B);
      MOVE(SIZE);
    END (*HIL2*);
  BEGIN (*HIL*)
    IF I=0 THEN TURN(180)
    ELSE
    BEGIN
      IF I>0 THEN
      BEGIN
        A:=90;  B:=I-1;
      END
      ELSE
        BEGIN
          A:=-90;  B:=I+1;
        END;
      HIL1;  HIL2;  HIL1;
    END;
  END (*HIL*);
BEGIN (*MAIN PROGRAM*);
  WRITE('SIZE:');
  READLN(SIZE); (*ENTER SIZE FOR YOUR SCREEN*)
  WRITE('ORDER:');  READLN(ORDER);
  PENCOLOR(NONE);
  N:=ORDER-1;
  DELTA:=SIZE;
  WHILE N>0 DO
  BEGIN  (*COMPUTE STARTING (X,Y) POSITION *)
    DELTA:=DELTA*2;
    N:=N-1;
  END;
  MOVETO(-DELTA,-DELTA);
  PENCOLOR(WHITE);
  HIL(ORDER);
END.
```

APPENDIX A - LIST OF COMMANDS

Command Character	Command Syntax	Command Semantics	Paragraph
A	A⟨command⟩	Iterate command by value of ACC	
B	B	Beep	
C	C	Clear screen	
D	D	Pen down	
E	E⟨then⟩⟨else⟩	Tests for turtle at screen edge	
F	F	Turtle forward one unit	
H	H	Turtle to home position	
L	L	Rotate turtle left 45°	
N	N	Point turtle north	
P	P	Value in ACC is pen color select	
R	R	Rotate turtle right 45°	
S	S	Color select in front of turtle to ACC	
T	T⟨then⟩⟨else⟩	Test ACC for non-zero	
U	U	Pen up	
W	W	Wait for next 30 HZ CLOCK TICK	
⟨blank⟩	⟨blank⟩	No-op	
!	!	Stop innermost iteration	
#	#⟨variable⟩⟨command⟩	Iterate command by value of variable	
$	$⟨select⟩⟨then⟩⟨else⟩	Test selected joystick position	
%	%⟨select⟩	Read selected pot controller to ACC	
&	&⟨select⟩	Value in ACC goes to selected color register	
(	(⟨command⟩....)	Binds command as a unit	
@	@	Set ACC to zero	
;	;	Sense orientation to ACC	

APPENDIX A - LIST OF COMMANDS
(continued)

Command Character	Command Syntax	Command Semantics	Paragraph
+	+	ACC = ACC + 1	
-	-	ACC = ACC - 1	
⟨number⟩	⟨number⟩⟨command⟩	Iterates ⟨command⟩ by value of ⟨number⟩	
=	=⟨name⟩⟨command⟩	Defines user command	
	= # ⟨name⟩	Defines user variable = ACC value	
?	? ⟨then⟩⟨else⟩	Random test	
[	[⟨command⟩...]	Same as () except ACC saved & restored	
—	—	No-op	
∧	∧	Increment current iteration count	
CTRL-A	⟨option⟩	Audio select	
CTRL-C		Clear (remove) all user variables	
CTRL-D	⟨mode⟩	Select display mode	
CTRL-E	⟨rule⟩	Select edge rule	
CTRL-G	"⟨device⟩"	Get user definitions from device	
CTRL-L	⟨name⟩	Load user definitions from ROM	
CTRL-M	⟨mode⟩	Select operating mode	
CTRL-P	"⟨device⟩"	Put user definitions to device	
CTRL-R	⟨name⟩	Load & run user command from ROM	
CTRL-S	⟨option⟩	Command execution speed select	
CTRL-T	⟨option⟩	Select turtle representation	
CTRL-Z		most	

APPENDIX B - ERROR STATUS CODES

A - Operator Abort (BREAK key)

D - Device name error

F - User command definition area full

I - Systems I/O error

L - Load/Run argument undefined

N - Nesting error (unmatched right bracket)

O - Overlength command line input

P - Incomplete (partial) line input

R - Reserved name for user command

S - Stack overflow

U - Undefined user variable name used

APPENDIX D - PEN SELECTS (BY SCREEN MODE)

Mode	Pen Range	Color Registers Used
0	0 - 127	
1	0 - 127	
2	0 - 3	0 = background 1 = PF0 2 = PF1 3 = PF2
3	0 - 1	0 = background 1 = PF0
4	0 - 3	0 = background 1 = PF0 2 = PF1 3 = PF2
5	0 - 1	0 = background 1 = PF0
6	0 - 3	0 = background 1 = PF0 2 = PF1 3 = PF2
7	0 - 1	0 = PF2 1 PF1 (luminescence only)

&0 = PF0
&1 = PF1
&2 = PF2
&3 = PF3
(&4 = background -- someday)

THE TURTLE

APPENDIX E - COLOR REGISTER VALUES

Range	Mid-Bright	Color
0 - 15	8	
16 - 31	24	
32 - 47	40	
48 - 63	56	
64 - 79	72	
80 - 95	88	
96 - 111	104	
112 - 127	120	
128 - 143	136	
144 - 159	152	
160 - 175	168	
176 - 191	184	
192 - 207	200	
208 - 223	216	
224 - 239	232	
240 - 255	248	

APPENDIX F - JOYSTICK TRIGGER TESTS

Select Code	Direction* Trigger	Device	Select Code	Direction Trigger	Device
A	F	JOYSTICK 0	U	T	POT 4
B	R	JOYSTICK 0	V	T	POT 5
C	B	JOYSTICK 0	W	T	POT 6
D	L	JOYSTICK 0	X	T	POT 7
E	F	JOYSTICK 1	Y	T	JOYSTICK 0
F	R	JOYSTICK 1	Z	T	JOYSTICK 1
G	B	JOYSTICK 1	[	T	JOYSTICK 2
H	L	JOYSTICK 1	\	T	JOYSTICK 3
I	F	JOYSTICK 2			
J	R	JOYSTICK 2	* F = Forward		
K	B	JOYSTICK 2	R = Right		
L	L	JOYSTICK 2	B = Backward		
			L = Left		
M	F	JOYSTICK 3	T = Trigger		
N	R	JOYSTICK 3			
O	B	JOYSTICK 3			
P	L	JOYSTICK 3			
Q	T	POT 0			
R	T	POT 1			
S	T	POT 2			
T	T	POT 3			
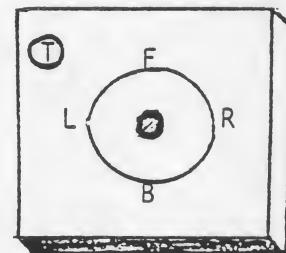
APPENDIX G - ROM RESIDENT USER COMMANDS   (See p.15)

Load/Run Name	Content	Runs	Reference
1	Y = SIERPINSKI CURVE J = HILBERT CURVE	Y	See 4.8, 4.9
2	Y = TRINARY TREE J = SIMPLE SPIRAL	Y	See 4.10, 4.11
3	K = SUPER SPIRAL	K	See 4.12
4	Y = DRAW (CARTESIAN) J = DRAW (POLAR)	Y	
5	Y = WALL BANGER J = Pi	Y	
6	J = HOLLYWOOD SQUARES	J	
7	Y = KOCH CURVE	Y	See 4.17
8	Y = THE ZAPPER	Y	FROM "ATARI WSFN" MANUAL
9	Y = DRAW	Y	
A	Y = POSIES	Y	
B	J = SUPERTURTLE	J	
C	COLORPOWER		
D	Y = MAGIC CARPET	Y	

APPENDIX H - AUDIO SELECT OPTIONS

Ø - audio off

A - LSB of horizontal position } relative to upper left corner

B - LSB of vertical position

C - MSB of software stack pointer

D - Pot controller Ø

E - Pot controller 1

F - value in CHAR (current command)

G - (internal)- TEMP

H - (internal)- TEMP + 1

I - (internal)- COUNT

J - hardware stack pointer

K - MSB of horizontal position } relative to screen center

L - MSB of vertical position

M - LSB of software stack pointer

N - (internal)- INPT +2

O - LSB of turtle accumulator

ATARI

WSFN

(TURTLE)

AN INTRODUCTION.....

ATARI WSFN INTRODUCTORY MANUAL

Prepared by: Gregory Yob

PREFACE   (General remarks that are unlikely to appear in the final
          version of this manual.)

    1) ATARI WSFN is subject to changes which may require changes
        to this manual- for example, the representation of the
        turtle might change.

    2) It is assumed that the user is familiar with the ATARI
        keyboard, and knows how to manipulate the SHIFT and CTRL
        keys.

    3) Entry of keys will be as typed with these exceptions:

            Letter ^{C} means CTRL-Letter

            Letter ^{S} means SHIFT-Letter

        For example, M^C2T^C1HCN is equivalent to:

            CTRL-M 2 CTRL-T 1 H C N

WHAT'S WSFN ????

        Have you ever tried to build a house out of toothpicks? Or
to bail out a battleship with a teaspoon? That's the situation
that the first computer hobbyists were faced with......

        Back in the "old days" of personal computers, you would buy
some kits through the mail, and with a soldering iron, a lot of
patience, and even more luck, you would end up with your very own
personal computer! However, that was only half the story - for
a computer without a program is like a car out of gas - or your
ATARI 400 (or ATARI 800) without any cartridges. You could turn
your computer on - and it would sit there and do nothing.........

        Deep inside every computer is the electronic equivalent of a
blackboard, and written on the blackboard is a series of instructions
for the computer to follow. You might imagine an ordinary blackboard
with a gridwork of 1/2 inch squares on it - if you filled some squares
with chalk, and erased others, you can draw letters - and then
words and sentences. Of course it takes hundreds of little squares
to write a short sentence......

When a computer is turned on, it looks at its memory (that's
the blackboard), and the pattern of ones and zeroes (white or black
squares) contains the instructions that tell it to do something. Since
it is easier and cheaper to build computers that recognize "binary
patterns" that to make computers that understand English sentences,
the computer's memory will look very strange to you and me.

Now, back in the "old days", the people with big computers had
already solved this problem - patterns of ones and zeroes were made
which let humans use typewriters and English-like sentences to tell
the computer what to do. (This is called programming.) But!! The
poor fellow with his personal computer didn't have these computer
languages - and his personal computer's memory was very small. In
fact, the memory was too small to hold the languages that big
computers were used to.

Computer hobbyists soon got tired of flipping switches
thousands of times to make their programs - so "tiny" languages
were invented. A tiny language couldn't do as much as the
languages on big comuters, but you can still do a lot more in
a tiny language, and do it a lot faster, than making up the
ones and zeroes by hand.

In 1977, Lichen Wang, who had already made a Tiny BASIC,
decided to make a tiny language for controlling robots with his
computer. When he shared it with his friends, they started to
call it different things - since Lichen hadn't given the language
a name. When Lichen was asked what to call the language, he
said: "If everything has to have an acronym, I would rather
call it WSFN (Which Stands For Nothing)."

When Lichen had WSFN ready, he didn't have a robot - but
he did have a TV set, and WSFN could be made to draw pictures
with the Turtle. (You can think of the Turtle as a video robot
if you want to.) Well, lets plug in our ATARI 400 (or ATARI 800),
put in the WSFN cartridge, and see what turtles can do.

THE TURTLE

Getting Started

Plug in the WSFN cartridge, and turn on the TV and your ATARI
400 (or ATARI 800). The TV should look like this:

```
┌─────────────────────────────────────────────┐
│                                               │
│                                               │
│                                               │
│                                               │
│                    ▄▄                         │
│                                               │
│              Turtle area (black)              │
│                                               │
│                                               │
├─────────────────────────────────────────────┤
│ ▟                  Entry area (blue)          │
└─────────────────────────────────────────────┘
```
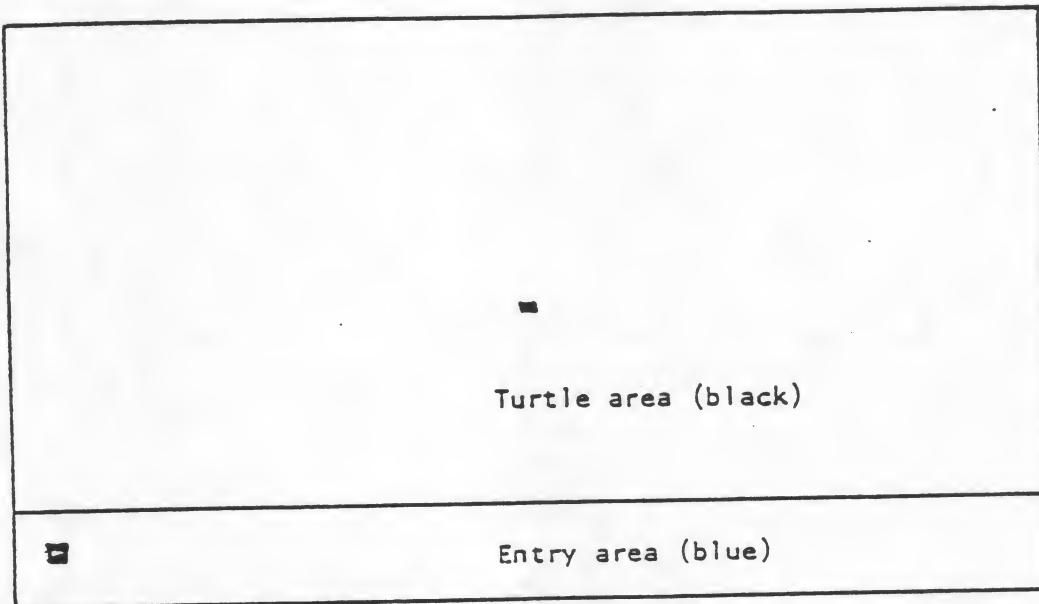
WSFN when the power is turned on.

The top part of the screen will be black, with a bright dot in
the center. The bottom part will be light blue, with a bright square
near the left side.

The top half is the world of the turtle, and here is where pictures
can be made. To draw pictures, you must enter commands with the keyboard,
and the blue area will show your last entry and the line you are entering.
(Nothing is shown when you start since nothing has been entered.)

How to See the Turtle

When WSFN starts up, the turtle is the bright dot - which doesn't
look like a turtle! To see a better turtle, enter on the keyboard:

CTRL-T 2

Press the CTRL key, and while it is down, press the T key. Then press
the 2 key (after releasing the CTRL key.)

NOTE: Instead of spelling SHIFT or CTRL for keystrokes, we will
use s or c like this: M^s V^c - these mean SHIFT-M and CTRL-V. The
turtle command shown above becomes T^c 2 .

— 3 —

Now the screen will look like this:
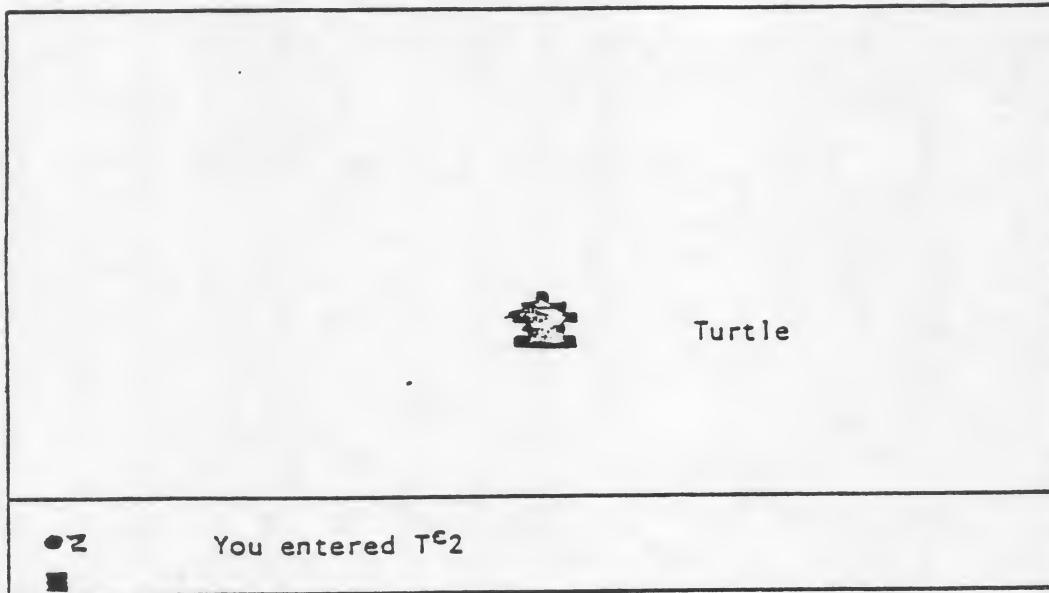
Making the Turtle appear.

The little animal on the screen is the turtle. If you look in the blue area, you will see a ● 2. The T^c is a graphics character,●, so this represents the command you just typed in.

(If you have used a computer before, an unusual thing is going on here: WSFN DOES A COMMAND AS SOON AS IT RECOGNIZES IT. You don't have to press RETURN - WSFN is awares of what you type in as each key is pressed.)

All About Turtles

Suppose you had a trained turtle, and you put him in the living room on a piece of paper:
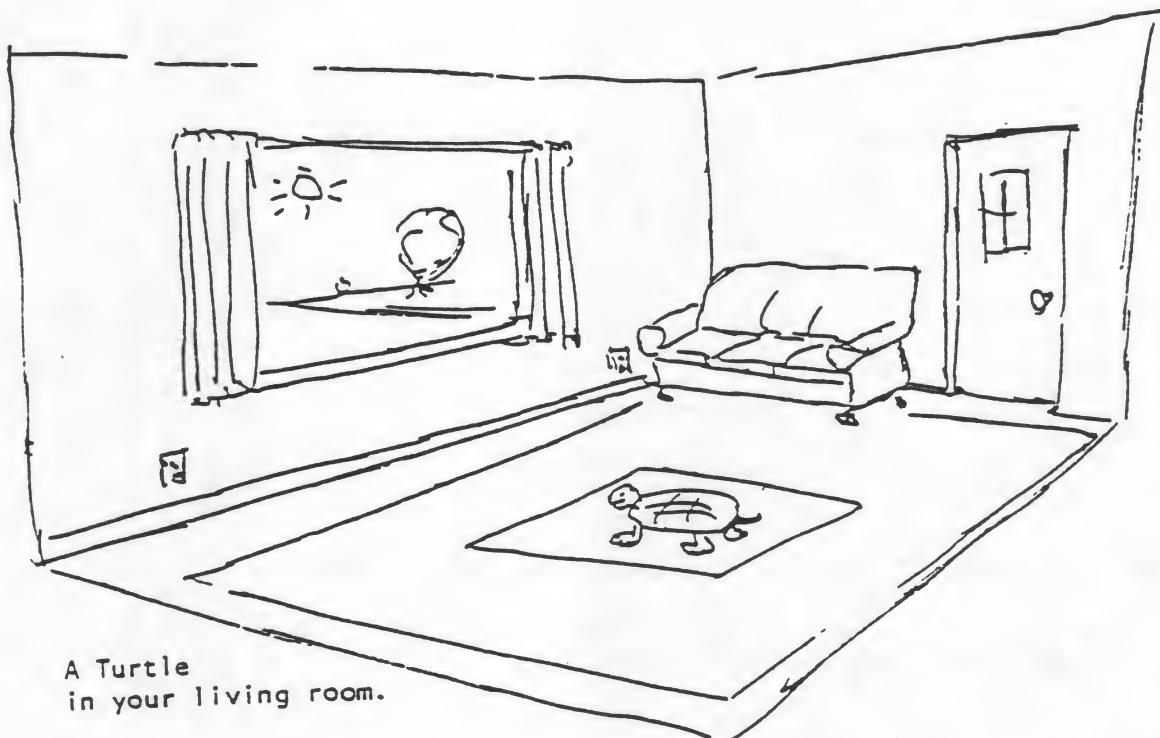
A Turtle
in your living room.

The turtle holds a felt pen in its mouth like this:
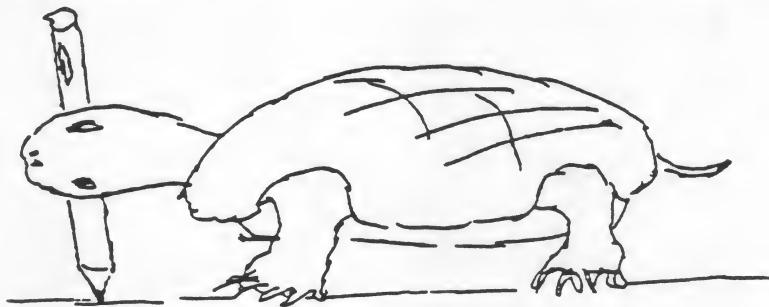
The pen is down.
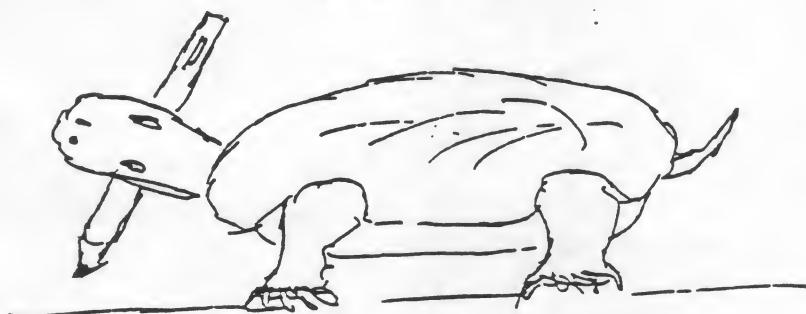
The pen is up.

If the turtle walked forward, the felt pen would drag along the floor,
leaving a mark on the paper:

Mark made by
pen
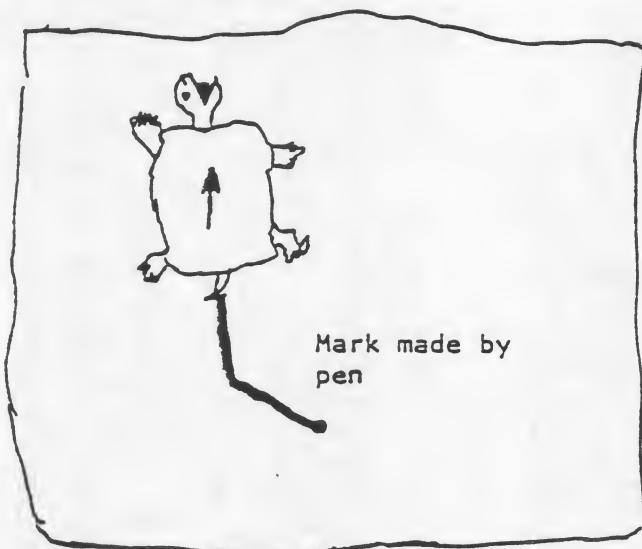
Turtle goes for a walk with the
pen down.

No mark would be made if the turtle was told to lift its head first.

Since turtles aren't too smart, you can only make him turn to
the right or the left by half- turns like this:

Left   Right
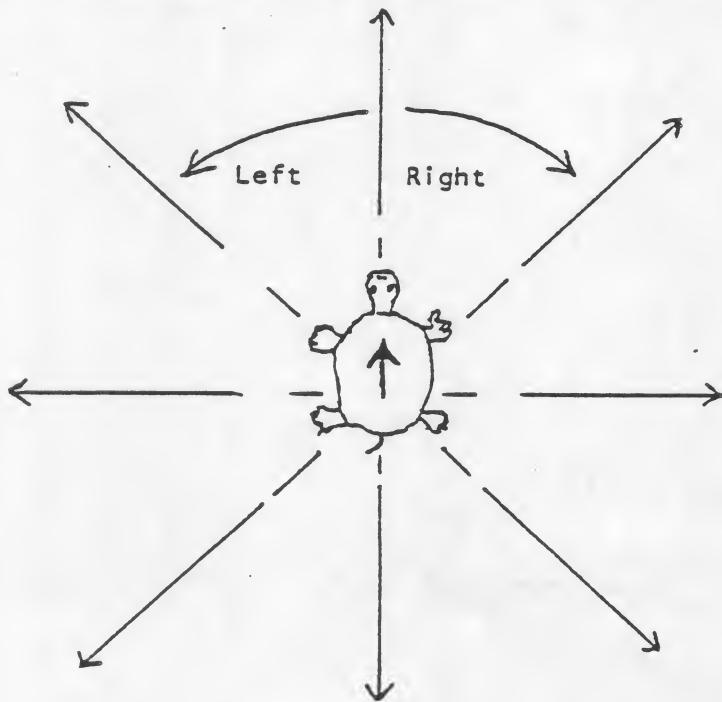
How a Turtle changes directions.

Sometimes the turtle might walk off the paper - and you'll have to pick him up and put him back in the middle. If you want, you can tell the turtle to point towards the living room window......

And, of course, when a drawing is finished, it's time to put a clean sheet of paper on the floor.

The WSFN turtle can do all of these things. The commands are:

U - UP   Lift the pen off the floor.

D - DOWN   Put the pen on the floor.

F - FORWARD   Move one step forward.

R - RIGHT   Turn right one half-turn.

L - LEFT   Turn left one half-turn.

H - HOME   Put the turtle in the middle of the screen.

N - NORTH   Point the turtle North (up on the screen).

C - CLEAR   Clear the screen.

Using the WSFN Turtle

Press the F key (FORWARD) a few times - if you do it enough, the turtle will move upward and a line appears beneath him:
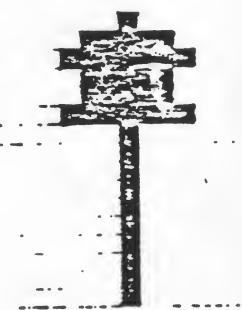
(Here you entered FFFFFFFFFFFFFFF.)

If you hold the F key down for a while, the ATARI will repeat the key (with beeps for each repeat.). The turtle will travel up the screen and disappear off the top edge.

Poof!

← Home is here

F
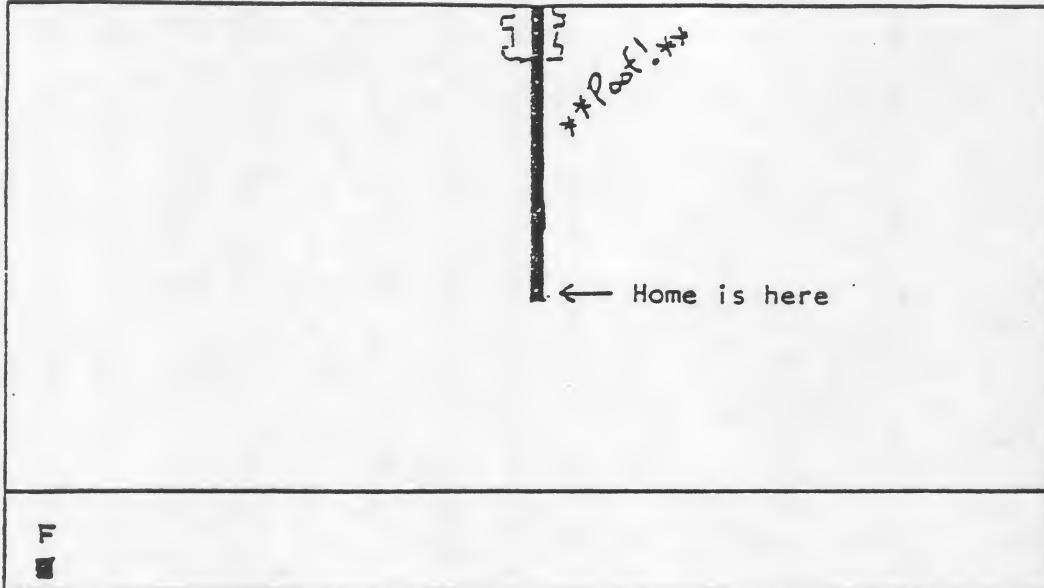
The Turtle walked off the top of the screen.

To get him back, use H (HOME) - and the turtle will reappear in the center again.

(Each time you press a turtle command key, WSFN does it immediately. This leaves only one letter as the last command shown in the bottom area on the screen.)

Now press R (RIGHT). The turtle will rotate a half-turn to the right. If you hold R down, the turtle will spin like a top! (Try it and see.)

Pressing L (LEFT) turns the turtle to the left by a half-turn.

Here is a challenge for you: See if you can draw the picture shown below:

- ON NEXT PAGE -
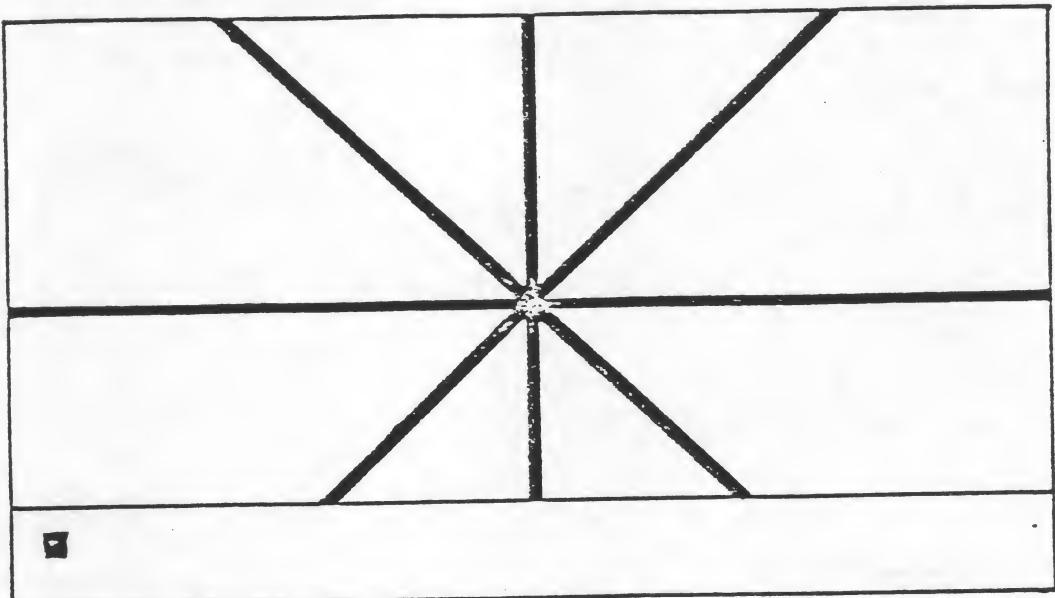
Can you draw this picture?

(Note: Diagonal lines will look like on the screen.)

(When the turtle goes into the blue area, the turtle is visible, but the line it draws isn't shown.)

To erase the screen, press C (CLEAR). The picture goes away, with the turtle remaining where it was before. Clear the screen and see if you can draw this picture:
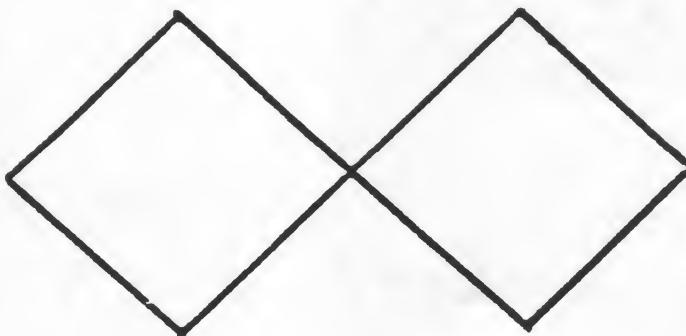
Another challenge for the drawing expert - and a small mystery.

How did the turtle get away from the drawing without a line??
Simple! He lifted his pen up - if you press U (UP), the turtle can
be moved without making a line. Now see if you can do the drawing and
leave the turtle in the spot shown.

(Hint: If you have a hard time making the lines meet in the center,
just count the number of F's you entered - and then it's easy. Counting
is very important in WSFN.)

To make the turtle draw again, use D (DOWN), and the pen will now
make a line on the screen.

Make the turtle spin around by holding down the R key. Then press
N (NORTH) and see what happens. Try this several times.

Some Optional Exercises

If you have a few moments, and want to really know your WSFN,
try the exercises in this section. Some of them aren't easy, so
feel free to skip them if you want to go on.

1) Draw a border around the edge of the screen. Keep count so you
know how many "footsteps" high and wide the screen is.

2) Draw a square 15 steps on a side, and fill it in by walking
around inside in a spiral.

3) Draw this figure:
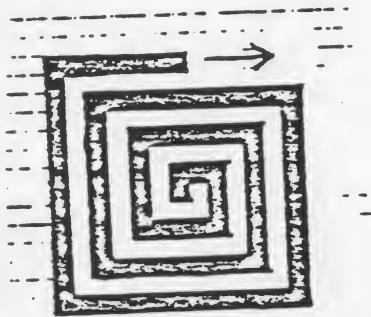
Spend some time making pictures with the turtle, and then go
on with the next section.

GETTING FANCIER

Suppose you wanted to draw a checkerboard on the screen. If you draw it one step at a time by pressing F, R and L, it will take a lot of effort! WSFN has several ways to save effort - so let's look at a few of these.

Repetition

If you enter a number before any WSFN command, WSFN will repeat the command. For example, clear the screen, home the turtle, point north, and enter:

25F

The turtle will move 25 units forward. (Be sure not to put a blank between the 25 and the F. A blank is a WSFN command to do nothing, and if you enter 25(blank), WSFN will do nothing - 25 times!)

Making a square is easy! Try entering:

HCN25F2R25F2R25F2R25F

If you did that right, you will see:
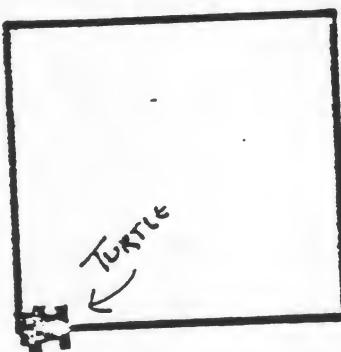
To twirl the turtle, enter:

9999R

(NOTE: 9999 is the largest number that WSFN understands. If you enter a bigger number, WSFN will take the four rightmost digits as the repetition value.)

Parenthesis

When you drew the square, did you notice that a square could be ... made by repeating this command four times?

four of: 25F2R

In WSFN, groups of commands can be put inside pairs of parenthesis and then repeated just like one command. Let's draw that square again:

HCN   (set up the screen)

4(25F2R)   (make the square)

If you look carefully at the screen, the turtle is now pointed up instead of pointing to the left. Why? (Hint: The first time you made the square, was there a 2R at the end?)

The blue area has another surprise - the entire sequence was entered, and _then_ WSFN did the command. WSFN didn't do each step as you typed them in.

When you enter a parenthesis, WSFN will wait until you enter the other parenthesis before doing the command. When you are doing a long set of commands, a parenthesis will let you see the entire line before WSFN executes it - if you make a mistake, the BACK S key can be used to erase letters. Let's try an example:

(The object is to make a rectangle 20 units high and 10 units wide.)

HCN

(20F2R10F2R20F3R10F   oops! you want 2R last time
          ↑

enter 5 BACK-S , and then enter:

2R10F)

No! Made 1 only...

(As an exercise, take a look at the picture that would result if you didn't correct the error - that's a mighty strange rectangle!)

Nesting Parenthesis

Pairs of parenthesis can be used to build bigger and better commands out of smaller ones. WSFN will wait until all the parenthesis are "balanced" - that is, the number of) matches the number of (. For example, try:

((((((((((20F))))))))))

The 20F won't be done until the last) is entered.

Here is a nicer kind of square:

8(4(20F2R)R)

Once this picture is on your screen (You have to actually <u>do</u> this one - we won't put the picture in this manual!), let's see if it can be figured out.

The way to understand lines with parenthesis is to work from the inside out, starting with the innermost pair. In this case, this is:

(20F2R)

This draws one side of a square, and turns the turtle 90° to the right. Now lets build up the next layer:

4(20F2R)R

The first part draws the four sides of the square. Then the turtle is turned once to the right. This means that the next time a square is drawn, it will be tilted diagonally.

Now, the entire thing is done eight times:

8(4(20F2R)R)

If you trace this out by hand, the turtle ends up at the Home spot after each square - but the turtle is turned once to the right each time. When the square was drawn the first time, the home spot is the lower left corner.....

WSFN tends to draw its pictures rather rapidly - to see how a picture is constructed maeans you have to slow WSFN down in some manner. The W command tells WSFN to wait for 1/30 second. Let's add this to the fancy square and see how the square is made:

8(4(20F2R30W)R)

After each side of a square is drawn, the turtle waits for one second - and it's much easier to see how the picture is made!

Just for fun, try these lines, and see if you can figure out how they work. If they go too fast, insert a 30W inside the innermost parenthesis (like we did with the fancy square).

                8(8(10FR)R)   (stained glass window)
                8(8(5FR)20FR)   (super eight)
                8(8(5FR)20FL)   (different for such a small change!)
                8(6(5FR)R)
                8(7(5FR)R)

You can see that small changes can make big differences! Here is a really long line to try:

                8(8(8(6(5FR)10FL)15FR)20FL)

Some Optional Exercises

Try the first few of these to see how complicated shapes can be made from simpler ones. It is quite difficult to type all of these in correctly, so feel free to quit when you get frustrated.

The series of entries shown below shows how to construct a checkerboard by making a square, and then adding more and more until a checkerboard is drawn. In the process, some mistakes (which computer programmers call bugs) will appear - and then they will be fixed. If you try to make complicated pictures of your own, you will make mistakes too.

The new parts of each entry will be underlined.

    1)   4(8F2R)   This draws a square

    2)   (4(8F2R)8F)   Once a square is made, move the
                             turtle to start a new square.

    3)   8(4(8F2R)8F)   Do eight squares in a row.

    4)   (HCN40F4R8(4(8F2R)8F))

                             The eight squares ran off the top of
                             the screen, so put the turtle near the
                             top and point him downward. Then do the
                             row of squares.

    5)   (HCN40F4R8(4(8F2R)8F)2R16F2R64F)

                             After doing a row of squares, put the
                             turtle in position to start a new row.

    6)   (HCN40F2R32F2R8(4(8F2R)8F)2R16F2R64F)

                             Just putting the turtle over to the
                             right to center the checkerboard on the
                             screen.

7)   ((HCNU40F2R32F2RD)8(4(8F2R)8F)(U2R16F2R64FD))

> When the turtle is being put into position,
> it shouldn't draw a line. The pen is lifted
> before a move and put down afterward. The
> new parenthesis separate the positioning
> moves from the square drawing and help make
> the entry easier to read.

(Note: If you have got this far, your entry will appear on two
lines of the screen's lower area. WSFN can accept up to
character lines.)

8)   ((HCNU40F2R32F2RD)8(8(4(8F2R)8F)(U2R16F2R64FD)))

> To make a checkerboard, just repeat the
> row of squares and the move to the next
> row eight times - but it doesn't work!
> (When a row is started, the turtle must
> point down, and we left it pointing up.)

9)   ((HCNU40F2R32F2RD)8(8(4(8F2R)8F)(U2R16F2R64F4RD)))

> With the turtle's direction fixed, we
> get a checkerboard, but with the rows
> too far apart - a new row starts 8 steps
> to the left instead of 16. Let's fix
> this and try again.

10)   ((HCNU40F2R32F2RD)8(8(4(8F2R)8F)(U2R8F2R64F4RD)))

> Wow! It works this time!!!!

MAKING YOUR OWN COMMANDS

Each time you want to draw a small picture, you have to enter the commands for the turtle - for example, to make three squares means you have to repeat:

    4(12F2R)   (draw the square)

    -------   (move the turtle to the next spot)

three times - once for each time the square is drawn. A design with a lot of squares could take a lot of typing!

WSFN lets you make new commands - and it will remember the command as long as the ATARI is turned on (or until you change the command's meaning). Here's how to make a "Square" command:

    =*S4(12F2R)

The = tells WSFN that a new command is being named. The name of the command is *S, and the meaning of the command is 4(12F2R) which draws a square.

Drawing the square is easy now - just enter:

    *S

and the square appears. You can also put your new command into other WSFN lines - for example:

    8(*SR)

    4(8(*SR)24F2R)

If you have been doing the examples, you surely have entered this one a lot:

    HCN

Why not make it a command as well - *C, for example:

    =*C(HCN)

You can include commands that you have invented into other commands. Let's take that fancy design and turn it into a command:

    =*F(*C4(8(*SR)24F2R))

Now by pressing just two keys, a complete picture is drawn!

(Notes: 1) WSFN uses the * to tell if a command is already in the "canned" commands that it already knows or if you invented it. 2) An invented command will end in the same way a normal WSFN command does - for example, if you want *Z to mean 4F3R, you should put some parenthesis around it like this: =*Z(4F3R) If you don't, WSFN will think that *Z means 4F. 3) WSFN doesn't actually do the command until you tell it to - the = is a command to record new commands. Use the command (such as *S or *Z) to make WSFN perform.)

A Command That Lasts Forever (Almost)

Let's make ≠Q a command with its own name included (This should drive WSFN nuts!):

=≠Q(≠SR4F≠Q)

The turtle draws the design below, and then does it over, and over, and over, and ......
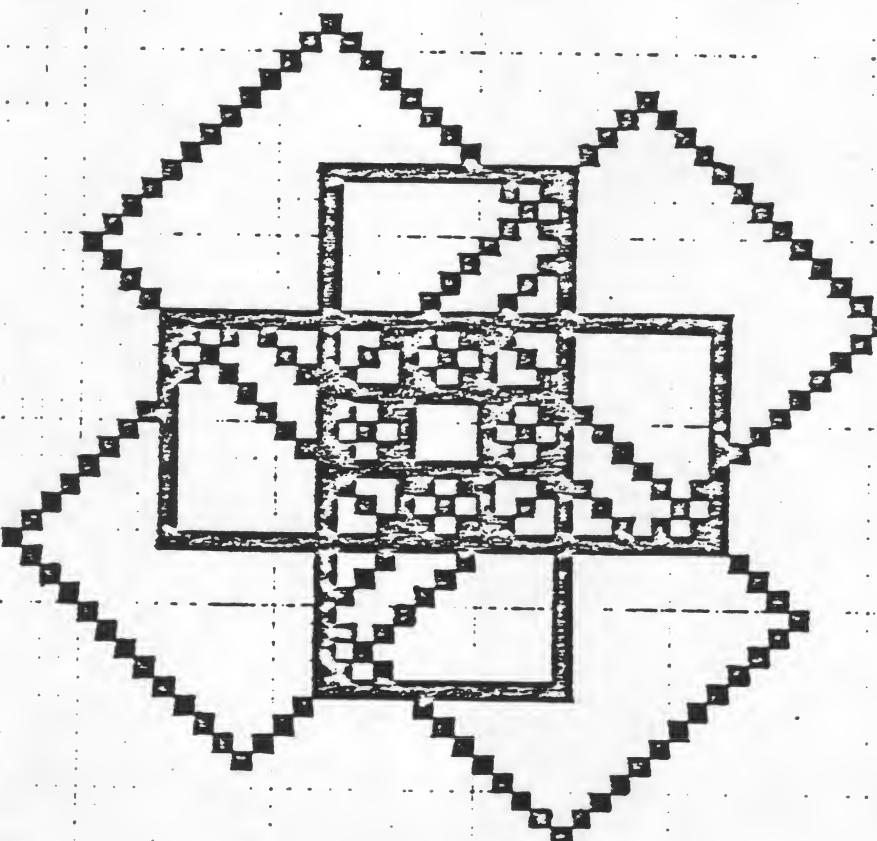
The command ≠Q tells WSFN to draw the square ≠S, move the turtle and turn it a little , and then, do ≠Q again! This is called an infinite loop - the same thing gets done forever....

(Note: Actually, WSFN will do *Q for several minutes, and then
quit. The reasons for this are beyond the scope of this manual. It is
possible to make true infinite loops in WSFN - again, using features
that aren't covered here.)

To make WSFN stop, press the BREAK key - BREAK will always stop
WSFN no matter what it is doing.

Enter *Q again, and while it is going, press the R or L keys
now and then - the figure will move around. WSFN will stop in the
middle of a command to do one that you just typed in (in this case,
R or L), and then continue doing the old command.

Looking at Your Commands

Now that we have several new commands, how do we look at them
if changes are needed. Just enter:

M^{C}1   (That's CTRL-M and 1)

The screen will change to:

```
ACC=0000  NUMBER=0000  LEVEL=0000
CHAR=1

>

■1



S=4(12F2R)
C=(HCN)
F=(*C4(8(*ST)24F2R))
Q=(5R4F*Q)
```

The Debug Mode Screen

— 18 —

This is called the "debugging mode". The four lines in the middle are the four commands you entered.

If you want to change a command, you can either type it in again, or use the cursor movements & screen editor to change the command's line directly. (This manual assumes that you know how to use the screen editor.)

There is one big difference in debugging mode - to make WSFN do a line, you must press the RETURN key.

To leave debugging mode, enter:

M^{c}3 RETURN   (CTRL-M, 3, and press RETURN)

(Note: WSFN's debugging mode has many features which won't be discussed in this manual. For the present, just use debugging mode to see your library of special commands, and to make changes to your special commands.)

THE ACCUMULATOR

WSFN has an "accumulator" - which is a counter which can be
used to keep track of what you are doing. For example, suppose
you want to draw a spiral square like this:
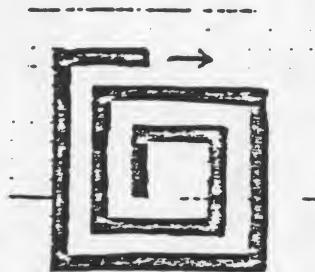
Each side of the square is one step larger than the previous one.
If you try to do this with the WSFN you already know, it gets harder
and harder to do:

       (3F2R)   (one side done)

       ((3F2R)F(3F2R))   (two sides done)

       =*S(3F2R)   (make a side a special
                                    command)
       (*SF*S2F*S3F*S4F)   (once around)

       (*SF*S2F*S3F*S4F*S5F*S6F*S7F*S8F)

       etc ....

The accumulator can be used to solve this problem. In WSFN,
the commands:

        +   adds one to the accumulator.

        -     subtracts one from the accumulator.

        A   repeats the next command as many times as indicated
               by the number in the accumulator.

Let's see how this works in practice. Starting with the turtle
at home, and pointing north, enter:

       ++++++++++   (press + ten times)

       AF

The turtle will move up 10 steps - that's the number in the
accumulator. If you repeat:

AF

the turtle will move another 10 steps forward.

Try entering several -, and then the AF. Now the turtle will move a shorter distance. If you press - enough times, the accumulator will have a zero in it. Now the turtle won't move at all.

The smallest number in the accumulator is zero (and the largest is 9999). If you enter too many -, the accumulator will just stay at zero. Try pressing - 15 times, and then AF - the turtle stays still.

The commands + and - can be repeated like any others. For example,

15+   adds 15 to the accumulator

23-   subtracts 23 from the accumulator

If you use A to tell how many + or - to use, you will get:

A-   sets the accumulator to zero

A+   doubles the number in the accumulator

(Be sure you understand the A+ and A-. A- is used a lot to "make things ready" - it is easy to enter a command which leaves some junk in the accumulator - and when you try the command again, WSFN will do something different! (and usually not very nice).)

If you want to see what's in the accumulator, go into debug mode and play a bit with the + and - commands. The accumulator is shown in the line that says:

ACC=0000

in the upper left corner. + and - will change this value, for example,

++++++ RETURN

will make the accumulator look like this:

ACC=0007

(Note: M^{c}1 gets you into debug mode, and M^{c}3 gets you out.)

Now, back to the square spiral - each side is one unit longer than the previous one, so:

(AF2R+)   will draw a side A units long, and add one to the accumulator.

Here is the spiral:

HCNA-40(AF2R+)

HCN sets up the screen, A- removes any leftover junk, and 40(AF2R+)
draws 40 sides to the spiral.

To fill up the screen, enter:

120(AF2R+)

Note that the spiral continues from where it stopped before - we didn't
zero the accumulator or home the turtle!

Just for fun, try these variations on spirals - and see if you
understand what they do. (The really serious student is asked to
visualize the picture before trying them out.)

Squares:

```
40(4(AFRR)+)
40((4(AFRR))+)
40((4(AFRR))++)
40((4(AFRR))++RR)
```

It is useful to define this command to set up the screen between
pictures:

```
=≠R(HCNA-)
```

Octagons:

```
40((8(AFR))+)
40(AFR+)
40(7(AFR)+)
40(6(AFR)+R)
```

Remember that you can insert some delay if you put a 30W inside the
innermost parenthesis, for example,

```
40(4(AFRR30W)+)
```

Single Step Mode

Sometimes WSFN will do things much too fast, and you will need to have a closer look at what's happening. (If you are just learning WSFN, skip this part until later.)

There are two commands that can help you look more closely:

M^C2   This puts the upper part of the debug mode display into the blue area - if you try some of the previous examples, you can see the value of the accumulator change. The line called:

CHAR=_

displays the current character that WSFN is looking at.

If you enter M^C3 you will return to the usual mode. (If you want a screen without any blue area, use M^CØ.)

S^C1   This is single-step mode. WSFN will do one action each time you press CTRL-4. For example, enter:

10+   (get into debug mode first)

and the accumulator will increment by one each time you press 4^C.

S^C has some other options also:

     S^CØ   returns WSFN to normal speed

     S^C7   makes WSFN do about 1 action per second.

When you are in either debug mode (M^C2 or M^C1), WSFN will run more slowly - it has the job of writing the debug data on the screen, and this takes some time to do.

Try some of the simpler examples that you already understand with these new commands - and you will see how they can help you understand complicated commands.

TESTING AND DECISIONS

WSFN has several commands which will select between two
commands. For example, if the accumulator is zero, one command
will be performed. If the accumulator isn't zero, the other
command will be done. These commands are known as tests, and
they all look like:

        (test something) (if the result is true, do this one)
                         (if the result is false, do this one instead)

The test for the accumulator looks like this:

        T(10F)(10R)

If the accumulator isn't zero, the first action (10F) is chosen. If
the accumulator is zero, the second action (10R) is done instead.
In both cases, the action that isn't chosen is ignored.

        Let's look at some of these in more detail:

The Random Test   (?)

        The command ? will select one of the two commands following it
with 1:1 odds (You can think of WSFN flipping a coin - if the coin
is heads, the first command is done, and if it is tails, the second
command is done.)

        Try the following entry on your ATARI - do it several times.
Each time you enter this command, the turtle either moves forward
5 units or turns to the right:

        ?5FR   (Remember that T^{C}2 sets up your turtle.)

        Repeating this can make some nice squiggles:

        1000?5FR

Note: If you can't get the ? to work, remember that ? is a shifted
character - you must press SHIFT to get a ?.

        Parenthesis can be used to build more complicated tests:

        100?(U5?5FR)(D8(5FR))

        Try it out, and then let's analyze what this expression does.
First, there is 100 repetitions of the expression:

— 24 —

?(U5?5FR)(D8(5FR))

The first action is U5?5FR which is selected ½ of the time. This lifts the pen, and moves the turtle randomly five times. (Just the previous examples with the pen up.)

The other action is D8(5FR) which draws an octagon on the screen.

Here are a few other random patterns to play with:

100?(10F?RL)(8(3FL)2R)
100?(H20?5F?RL)(2R8(2FR)2L)
100?(H20?5F?RL)(10?(2R8(4FL)2L)12F)

The Accumulator Test   (T)

The command T checks the accumulator. If the accumulator is not zero, the first action after the T is performed. If the accumulator is zero, the second action takes place.

By setting the accumulator, you can make your commands more responsive - they can check the accumulator and select different actions if the accumulator is zero.

Here is an example:

=*ET(20F2R)(4(5F2R))

HCNA-

Entering *E will draw a small square (That's the 4(5F2R) when the accumulator is zero.)

If you do a +, *E now moves the turtle 10 units and turns it right - doing this four times makes a larger square.

Counting the number in the accumulator can be handy - let's see what can be done with spirals:

=*ST(AF2R-*S)(sp)   (sp means SPACE)

If something is in the accumulator, draw part of the square spiral. If the accumulator is empty, do the sp - space is WSFN for "do nothing". Here it means we are done with the spiral.

After the side of the square is drawn, subtract one, and then call *S to do the next side - this will continue until the accumulator is zero.

Try setting the accumulator to different values and see the spirals that can be drawn.

If you want to check if the accumulator is larger than 32, use:

32-T(etc)

First, subtract 32, and then test the accumulator.

The Use of Brackets

If you enclose some WSFN commands in brackets [ ], an interesting thing happens - the accumulator is saved, and no matter what you do to the accumulator inside the brackets, it will be restored to its old value when you come outside. The brackets also work like parenthesis for grouping commands together.

Here is an example - we will move the turtle forward by the number in the accumulator, draw the spiral with *S, turn right, subtract 5 from the accumulator, and repeat until the accumulator is empty.

If we used parenthesis around *S, only one spiral would be drawn - why? - the command *S leaves the accumulator at zero. The cure is to put brackets around *S which preserves the accumulator while *S is performed. Let's try it:

(be sure *S is already entered)

=*QT(AF[*S]2R5-*Q)(sp)

To try this out, enter:

HCNA-30+*Q

It's fun to try different values of the accumulator and see what *Q does.

Some Last Optional Exercises

1) Change *S to some of the other spirals and octagons in the section on the accumulator.

2) Change *Q to move about randomly, and then to set the accumulator at random (from 2 to 20) and call *S - and do this 20 times.

3) Look at the optional exercises that draw a checkerboard. Break the expressions down into several * commands which A) draw the square, B) move the square one unit, C) draws a row of squares, D) moves to the next row, and E) does the entire board. Notice how much easier it is to check each part and to fix bugs (mistakes that you made, being a fallible human being).

SOME FINAL COMMENTS

Now that you have survived this introduction to WSFN, there's
a lot more to cover. (Get the reference manual for details.)
WSFN can:

1)   Check the joysticks position and triggers.
2)   Check the paddle pots and triggers.
3)   Make sounds - both beeps and via the TV set.
4)   Change the colors in the display.
5)   Check if the square in front of the turtle has been
     drawn on.
6)   Save and load your commands from discs and tapes.
7)   Load "canned" commands from the cartridge ROM.
8)   Control repetition in several other ways.
9)   Represent the turtle - or make it go away in different
     ways.
10) Control how the turtle treats the edge of the display.
11) Save the accumulator in "variables" for later use.
12) Report some errors in debug mode.

SOME DEMONSTRATION PROGRAMS

The following demonstrations illustrate some of the extra
features in WSFN and are fun to play with.

Please take care to see that you have entered them correctly -
an error in entry usually means the program won't work. Each group
is separate unless stated otherwise.

The Zapper

The Zapper illustrates the use of sound and color options
in WSFN. Three procedures, *A, *C, and *D are required.

```
=*A(M^CØA^CGA-1(+P&1*O))
=*C(&Ø[A+&1][T(AF*O&2-*O&3)H ])
=*D(3R2F2R)
```

Start the Zapper with *A.

Variations of the Zapper involve changes to *D:

```
=*D(?(3R2F2L)(3L2F3L))
=*D[??(AFR-)(L)]
=*D[T(7(AF2R)FR-)sp]                    sp means SPACE
```

Several of the shapes in the manual, such as the octagonal and square
spirals, can be used. Enclose these procedures in square brackets []
to preserve the accumulator.

Some Joystick Exercises

WSFN can look at the joysticks, and these examples show a few possibilities. The joystick is to be plugged into Port Ø (the left-most port in front.)

Each of these joystick programs are started with *J.

Draw Program

This makes the joystick like a turtle. UP moves the turtle forward, RIGHT turns the turtle R, LEFT turns the turtle L, and DOWN puts the turtle at home (H).

```
=*J(M^CØT^C21(*K∧))
=*K($A(FB)sp$B(RB)sp$D(LB)sp$C(H5B)sp)
```

Planting Posies

```
=*J(M^CØT^C2HN1(*K∧))
=*K(U$AFsp$BRsp$DLsp$C(D*PB)sp2W)
=*P8(8(3FR)3L6F)
```

Here, DOWN plants a "posie", and the other controls move the turtle around with the pen up. An alternate "posie" is:

```
=*P(8(8(3FRA^CAW)3L6F2RA^CØ))
```

and each "posie" makes a sound!

The Superturtle

Since the WSFN turtle has only 8 directions, here is a "high resolution" turtle with 24 directions. The commands *F, *R, and *L perform the Superturtle equivalents of F,R, and L. *F will move the turtle about 4 units horizontially/vertically and about 3 units diagonally. (Surely you have noticed that F on the diagonal is longer than F off the diagonal. *F tries to correct this distortion.)

The Superturtle commands will not change the accumulator and can be substituted into other figures. (ie, change F to *F, etc.)

D^C7M^CØ will set up a high resolution display (2 colors only) in which Superturtle can be used nicely.

```
=*U4F
=*V(2FRFLF)
=*W(FR2FLF)
=*X(R3FL)
=*Y(2RFL2FRF2L)
=*Z(2R2FLFRF2L)
```

*U to *Z draw in each Superturtle direction.

— 28 —
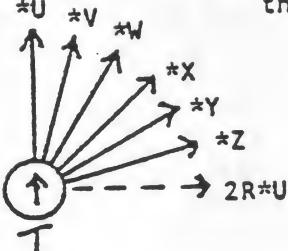
When the turtle is pointed North, the procedures *U to *Z will move as shown. The turtle remains pointed North. To move in the other directions involves using 2R or 2L to get into the next quadrant.

Superturtle Directions

The individual directions should be checked by homing the turtle and entering 8 or 10 repetitions for each line. For example, H8*V to check *V.

=*F[A-#Z+T(-T(-T(-T(-T(-Tsp*Z)*Y)*X)*W)*V)*U]
=*R[A-#Z+4-T(A-=#Z2R)(#Z++=#Z)]
=*L[A-#Z+T(-=#Z)(5+=#Z2L)]

If the six direction primitives are correct, this expression will draw a nice circle:

(4)—(6)(*U*V*W*X*Y*Z2R)   (Be sure Turtle is not on a diagonal.)

Here are some pictures to do with Superturtle - most are variations of the same idea:

    24(H*R12*F)
    24(H4(*R3*F)3*L)
    24(H12(*R*F)11*L)
    24(12*F11*R)
    (A-30(24(A*F*R)+*R))
    24(24(*F*L)*R)
    3(24(24(*F*L)*R)3*F)
    4(6(H12(*L*F)+P11*R))

The Superstick

If you have the Superturtle entered, here is a superjoystick which does the Superturtle's version of the draw program. *J will get things going.

    =*J(HCNMC∅1($B*Fsp$B*Rsp$B*Lsp$GHsp∧2W))

The Superstick can be "flown" quite nicely.

Colorpower Machine

Control of the color registers can create some nice effects. The procedures offered here permit the use of two joysticks to control the colors in a drawing.

```
=*A(A-#A++=#A&1)
=*B(A-#B++=#B&2)
=*C(A-#A+-=#A&1)
=*D(A-#B+-=#B&2)
=*E(A-#C++=#C&3)
=*F(A-#D++=#D&Ø)
=*G(A-#C+-=#C&3)
=*H(A-#D+-=#D&Ø)
```

Note: If WSFN had @1 mean load acc with color in color reg 1, these procedures would be much shorter. (ie, =*A(@1+&1) .)

A nice test figure is drawn by invoking *M:

```
=*M(MᶜØTᶜØHCNA-=#A=#B=#C=#D*N)
=*N8(3+P[32(8(AFR)+)]R)
```

If you enter the joystick control, invoke *M and then *J, the joysticks will change the colors. Opposite directions (UP - DOWN, RIGHT-LEFT) change a single color.

```
=*J1($A*Asp$B*Bsp$C*Csp$D*Dsp$E*Esp$F*Fsp$G*Gsp$H*Hsp3W∧)
```

The color control routines can make some nice effects by being included in other WSFN expressions. Change *N as follows, invoke with *M, and then try: (You might want to use Mᶜ3 in *M instead)

```
=*N(150?(+PH)(20(3F?RL)))
```

```
1(?(?*A*B)(?*C*D)W∧)
(A-1(A[*A]+A[*B]+A[*C]+A[*D]∧))
1(???????*A*B*C*D*E*F*G*Hspspspspsp∧)
1(∧?(?(?*A*B)(?*C*D))(?(?*E*F)(?*G*H)))
1(∧?(?(16*A)(16*B))(?(16*C)(16*D)6W))
```

NOTES FOR DEMONSTRATIONS

These brief notes will help explain how the various demonstrations operate and the special features used.

The Zapper: *A sets up the screen in full graphic mode (MᶜØ) and turns on the sound (AᶜG) and zeroes the accumulator. It then repetitively calls *C. The caret increments the iteration counter. When this is inside a parenthesis preceded by 1 or more, the result is a true infinite loop. The command P sets the turtle's pen color to the value in the accumulator - in WSFN, this is the 4 values selected by the 2 LSB of the accumulator.

*C plays with the color registers &0, &1, &2, and &3.
The & sets the indicated color register to the accumulator's
value. Each color register may have a value from 0 to 255, and
the color selected will appear on any lines drawn with the
corresponding pen selection (P). *C then moves the turtle
A units, invokes *D, and then invokes itself recursively. When
the accumulator zeroes, the turtle is homed, and *C is at last
finished.

*D is used to draw any convenient figure to embellish
the plot.

Joystick. Exercises   *J sets up the screen with a turtle and
repeatedly calls *K with the caret trick.

*K tests each position of the joystick and then the
turtle is moved appropriately (ie, FB, RB, etc). The beep
is included to provide feedback, and to slow things down to
human speed. Home gives a longer beep.

In the posies, *J is a little better with setup of the
screen.. *K lifts the pen and moves the turtle - except with
the DOWN option which puts the pen down and draws the posie.
The 2W is needed to slow things down.

The first *P is clear enough. The second one turns the
sound on, and then off. (A^C followed with A-P makes some "artifact"
sounds with the letter selecting the artifact. A^{C}0 turns the
sound off.. Try it with other WSFN drawings.)

$ followed by a letter tests a joystick position and
does the first command if the joystick is moved, and the
second if it is not. For example, $EFH would move the turtle
if the joystick is UP, and home the turtle if the joystick
wasn't UP.

The joystick positions are:
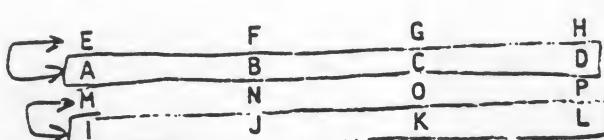
UP	RIGHT	DOWN	LEFT	
E	F	G	H	Port 0
A	B	C	D	Port 1
M	N	O	P	Port 2
I	J	K	L	Port 3

Superturtle: *U to *Z draw one of the 6 directions in each
quadrant. Turning the turtle 90 degrees permits re-use of these
procedures in the three other quadrants.

*F tests the value of a variable #Z which stores the
superturtle's "direction" which is from 0 to 5. It then
calls the appropriate drawer from *U-*Z.

*R checks #Z, and does one of:
        If #Z is 5, set #Z to zero and turn the turtle
            right 90 degrees.
        Else add 1 to #Z.

*L checks #Z, and does one of:

        IF #Z is zero, set Z to 5 and turn the turtle
            left 90 degrees.
        Else subtract one from #Z.

        WSFN permits the storage of the accumulator's value
into variables, identified by # (#A to #Z). The two operations
on variables are:

    =#var   Set var to value in accumulator
    A-#var+   Set accumulator to variable's value

Variables can also be used for repetitions (as shown in setting
the accumulator above.)

Superstick: Done by substituting the Superturtle *F, *R and *L
into the draw program and putting the loop & setup into *J also.

Colorpower Machine: Procedures *A through *H increment or
decrement the color registers. Variables #A to #D hold the
color register values.

        *M is used to set up the screen - and! to initially define
the variables used. *N draws an abstract pattern which has many
textures.

        *J is an extended version of the usual joystick text -
now handling two joysticks in Port Ø and Port 1. When running,
three of the colors will be changed. One set is the background,
and cannot change (at present). As the values in #A to #D
increase, the program runs more slowly due to the #A+ (etc)
needed to set up the accumulator. The colors in the color registers
are arranged in a way that changes brightness faster than hue
which takes getting used to.

        The various expressions randomly change the color values
by different ruses and will run indefinitely. *N was changed
for the heck of it.

LIMITED WARRANTY ON MEDIA AND HARDWARE ACCESSORIES.

We, Atari, Inc., guarantee to you, the original retail purchaser, that the medium on which the APX program is recorded and any hardware accessories sold by APX are free from defects for thirty days from the date of purchase. Any applicable implied warranties, including warranties of merchantability and fitness for a particular purpose, are also limited to thirty days from the date of purchase. Some states don't allow limitations on a warranty's period, so this limitation might not apply to you. If you discover such a defect within the thirty-day period, call APX for a Return Authorization Number, and then return the product along with proof of purchase date to APX. We will repair or replace the product at our option.

You void this warranty if the APX product: (1) has been misused or shows signs of excessive wear; (2) has been damaged by use with non-ATARI products; or (3) has been serviced or modified by anyone other than an Authorized ATARI Service Center. Incidental and consequential damages are not covered by this warranty or by any implied warranty. Some states don't allow exclusion of incidental or consequential damages, so this exclusion might not apply to you.

DISCLAIMER OF WARRANTY AND LIABILITY ON COMPUTER PROGRAMS.

Most APX programs have been written by people not employed by Atari, Inc. The programs we select for APX offer something of value that we want to make available to ATARI Home Computer owners. To offer these programs to the widest number of people economically, we don't put APX products through rigorous testing. Therefore, APX produts are sold "as is", and we do not guarantee them in any way. In particular, we make no warranty, express or implied, including warranties of merchantability and fitness for a particular purpose. We are not liable for any losses or damages of any kind that result from use of an APX product.

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many software authors are willing and eager to improve their programs if they know what users want. And, of course, we want to know about any bugs that slipped by us, so that the software author can fix them. We also want to know whether our documentation is meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program _____

2. If you have problems using the program, please describe them here.

_____

_____

_____

3. What do you especially like about this program?

_____

_____

_____

4. What do you think the program's weaknesses are?

_____

_____

_____

5. How can the catalog description be more accurate and/or comprehensive?

_____

_____

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program?

_____ Easy to use
_____ User-oriented (e.g., menus, prompts, clear language)
_____ Enjoyable
_____ Self-instructive
_____ Useful (non-game software)
_____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

_____

_____

_____

8. What did you especially like about the user instructions?

_____

_____

9. What revisions or additions would improve these instructions?

_____

_____

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

_____

_____

11. Other comments about the software or user instructions:

_____

_____

_____

_____

_____

_____

```
 -----
|     |
|STAMP|
|     |
 -----
```

ATARI Program Exchange
Attn: Publications Dept.
P.O. Box 50047
60 E. Plumeria Drive
San Jose, CA 95150

[seal here]